

# SPRINT PLANNER

Handle your releases and organize your team

## Table des matières

<b>INTRODUCTION</b>	4
COMPETENCES COUVERTES	5
ACTIVITE 1 - Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité	5
ACTIVITE 2 - Concevoir et développer la persistance des données en intégrant les recommandations de sécurité	5
ACTIVITE 3 - Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité	5
PROJECT SUMMARY	6
<b>EXPRESSION DE BESOIN ET GESTION DE PROJET</b>	7
DEFINITION DES TERMES	8
Termes liés à l'agilité	8
Termes liés à l'équipe	8
EXPRESSION DE BESOIN DU PROJET	9
Client	9
Définition du besoin	9
Analyse de l'existant	10
Solution préconisée	12
GESTION DE PROJET	13
Planning et suivi	13
Environnement humain et technique	14
Objectifs de qualité	15
<b>SPECIFICATIONS FONCTIONNELLES ET TECHNIQUES</b>	16
SPECIFICATIONS FONCTIONNELLES	17
Contexte	17
Acteurs	17
Fonctionnalités principales	19
SPECIFICATIONS TECHNIQUES	26
Ecosystème de données (stack technique)	26
Base de données et diagrammes	27
Règles d'implémentation	28
Bonnes pratiques	30
<b>IMPLEMENTATION ET TESTS</b>	33
IMPLEMENTATION	34
Organisation du code	34

Base de données .....	35
Sélection du candidat par l'algorithme de round robin.....	36
Couche Serveur .....	38
Couche de présentation.....	44
TESTS .....	48
Conventions .....	48
Configuration .....	48
Exemples d'implémentation .....	49
<b>VEILLE ET RECHERCHES</b> .....	52
VEILLE SUR LE SUJET DE LA SECURITE .....	53
Veille sur les failles XSS .....	53
Veille sur les failles CSRF .....	54
SITUATION DE TRAVAIL AYANT NECESSITE UNE RECHERCHE.....	54
Fonctionnement de JAVAMAIL .....	55
<b>CONCLUSION</b> .....	56
BILAN.....	57
Appréciation.....	57
Evolutions à venir.....	57
REMERCIEMENTS .....	58
BIBLIOGRAPHIE .....	58
<b>ANNEXES</b> .....	59
ANNEXE 1 - Diagrammes de classe .....	60
ANNEXE 2 - DDL et DML.....	62
Script de création de schéma.....	62
Script d'insertion de données .....	66

# INTRODUCTION

## COMPETENCES COUVERTES

ACTIVITE 1 - Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

CP 1	Maquetter une application	<input checked="" type="checkbox"/>
CP 2	Développer une interface utilisateur de type desktop	<input type="checkbox"/>
CP 3	Développer des composants d'accès aux données	<input checked="" type="checkbox"/>
CP 4	Développer la partie front-end d'une interface utilisateur web	<input checked="" type="checkbox"/>
CP 5	Développer la partie back-end d'une interface utilisateur web	<input checked="" type="checkbox"/>

ACTIVITE 2 - Concevoir et développer la persistance des données en intégrant les recommandations de sécurité

CP 6	Concevoir une base de données	<input checked="" type="checkbox"/>
CP 7	Mettre en place une base de données	<input checked="" type="checkbox"/>
CP 8	Développer des composants dans le langage d'une base de données	<input type="checkbox"/>

ACTIVITE 3 - Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité

CP 9	Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement	<input checked="" type="checkbox"/>
CP 10	Concevoir une application	<input checked="" type="checkbox"/>
CP 11	Développer des composants métier	<input checked="" type="checkbox"/>
CP 12	Construire une application organisée en couches	<input checked="" type="checkbox"/>
CP 13	Développer une application mobile	<input type="checkbox"/>
CP 14	Préparer et exécuter les plans de tests d'une application	<input checked="" type="checkbox"/>
CP 15	Préparer et exécuter le déploiement d'une application	<input checked="" type="checkbox"/>

## PROJECT SUMMARY

Every company working in Agile mode has many recurring **questions**:

**Who oversees the release this week?**

**Who oversees the support?**

**Did someone handle non-regression tests for the release?**

Each week, the releaser is determined manually by the team leader often lately and technical leaders and experts are almost every time designated by default. Support task are given to the very same technical leaders and experts.

This leads to a work overload and tiredness for those experts and to lack of knowledge sharing and skills improvement to other people in the team.

**Sprint-planner** application has been created **as an answer**.

Sprint-planner provides **a fair task sharing** by using a **round robin** algorithm to make sure everyone in the team is involved in releases and support. Everyone must handle the release and support. Therefore, everyone **improves their skills**. This also **alleviate expert's workload** making them available to develop features and help their teammates.

The application is **flexible**, so we can change the selected member manually if needed. Support is handled for both French and Indian team, and both technical and functional team.

Sprint-planner is built as an "Out-of-the-Box" application, so any team can use it simply by deploying it in any environment. It can be configured to fit with most of the Agile team requirements.

The idea is born from a request from the team, who asked for a tool sending a mail to a designated releaser each week. A thorough study of the workflow enlightened the need for **release, support and non-regression tests** to be handled. Another identified need was to list which development was delivered by **reconciliating GitHub and Jira issues**, two agile tools, to ease both releasers and business analysts work.

# EXPRESSION DE BESOIN ET GESTION DE PROJET

## DEFINITION DES TERMES

### Termes liés à l'agilité

#### Product Owner

Le product owner est le responsable de la définition et de la conception du produit. Expert de la méthodologie agile, il fait le lien entre la partie métier (client) et la partie technique (développeurs).

#### Agile Master

L'agile master est un coach agile. Il est garant des bonnes pratiques de l'agilité d'une équipe.

#### Agile@Scale

L'agilité à l'échelle (Agile@Scale) est une pratique de l'agilité sur plusieurs équipes qui travaillent sur des projets différents avec un objectif commun.

#### Feature team

L'équipe est organisée en deux sous équipes qui travaillent sur des fonctionnalités spécifiques à un sujet. Ces deux sous équipes sont appelées feature team.

#### P.I.

Dans le cadre de l'Agile@Scale, les différentes équipes organisent leur travail grâce au train de l'agilité. Ce dernier est représenté par des incréments de produit (Product Increment ou P.I.). Chaque P.I. contient un certain nombre de sprints (5 dans notre cas).

#### Sprint

En agilité, un sprint est une période définie (2 semaines dans notre cas) pendant laquelle l'équipe travaille sur des fonctionnalités définies en début de sprint et priorisées avec le Product Owner.

#### backlog

Le backlog contient tous les besoins client et est organisé par priorité. Il sert à effectuer les fonctionnalités par ordre de priorité et ainsi répondre au mieux au besoin client.

### Termes liés à l'équipe

#### Business Analyst

Les business analyst sont les maîtres d'ouvrage. Ils ont la charge de la compréhension et du bon respect des spécifications fonctionnelles données par le client.

#### Shift

Un shift, littéralement « décalage » représente une équipe géographique. L'équipe parisienne représente un shift, l'équipe indienne un second. Les deux équipes travaillent en décalage horaire, d'où l'emploi de ce terme.



## EXPRESSION DE BESOIN DU PROJET

### Client

L'équipe que j'ai intégré est le client de l'application puisque tous les membres seront les utilisateurs.

Le Product Owner sera l'Agile Master de l'équipe, puisqu'il est à la fois client, responsable fonctionnel et mon tuteur.

### Définition du besoin

Mon équipe souhaite bénéficier d'un outil qui permettra de répondre aux besoins suivants :

#### *Gestion d'équipe*

- Sélectionner un membre pour les tâches de déploiement, de support et de test de non-régression
- Traiter la sélection de façon automatisée et planifiée
- Modifier au besoin la personne sélectionnée (en cas d'absence par exemple)
- Sélectionner à nouveau une personne n'ayant pas pu accomplir sa tâche (en cas d'absence par exemple)
- Informer le membre sélectionné ainsi que l'équipe

#### *Assistance au déploiement*

- Effectuer une réconciliation entre les tickets Jira et GitHub afin de faciliter la rédaction de la release note<sup>1</sup>
- Choisir les branches GitHub à comparer pour effectuer la réconciliation

#### *Utilisateurs*

Les destinataires du projet sont les membres de mon équipe :

- Les développeurs de mon équipe
- Notre maîtrise d'ouvrage

#### Développeurs

Les développeurs de mon équipe sont au nombre de 24 séparés en deux shifts : 18 sont à Paris et 6 sont à Bangalore. Ils sont tous à l'aise avec l'outil informatique et les outils utilisés.

Cependant, le fait que l'équipe soit séparée en deux entre l'Inde et la France implique de prendre en compte la différence de fuseau horaire notamment pour la planification.

#### Business Analysts

Les Business Analysts sont les maîtrises d'ouvrage du projet. Ils sont 5 à Paris et 1 à Bangalore. Ils sont moins à l'aise avec l'outil informatique mais plus habitués à la gestion de projet et à l'utilisation de Jira.

---

<sup>1</sup> Note de déploiement recensant les développements embarqués, à destination du product owner et des sponsors de l'application

Il conviendra de prendre en compte cette différence de maîtrise dans le maquettage de l'application afin d'apporter un outil ergonomique et facile d'utilisation.

## *Rôles*

### Releaser

Est considéré comme releaser le membre qui prend en charge le déploiement de la nouvelle version de d'une application. Les déploiements ayant lieu chaque fin semaine, il est choisi à chaque début de semaine. Il peut ainsi suivre les développements de la semaine et savoir ce qui sera déployé.

### Support

Est appelé support le membre en charge des demandes utilisateur en cas de dysfonctionnement d'une application. Il prend en charge, analyse et fixe les anomalies. Cette tâche peut s'avérer chronophage tant dans l'analyse que dans la résolution des incidents.

Le support doit être assuré par les équipes de Paris et Bangalore, par un développeur et un Business Analyst.

### Tester

Est désigné testeur un membre de l'équipe en charge des tests de non-régression de l'application. La grande majorité des tests (unitaires, intégration, fonctionnels) sont exécutés automatiquement mais les tests de non-régression et de non-régression sont effectués manuellement en amont des déploiements.

## Analyse de l'existant

### *Dans l'entreprise*

#### Gestion d'équipe

L'équipe détermine aujourd'hui la personne en charge du déploiement chaque semaine. L'Agile Master décide du membre qui s'occupera de cette tâche.

Le support incombe à celui qui est disponible quand un incident survient.

#### - Avantages :

La sélection est flexible et un membre spécifique peut être sélectionné dans certains cas (un tech lead<sup>2</sup> pour une livraison sensible par exemple)

L'Agile Master décidant de qui sera sélectionné, il est toujours informé du releaser en cours.

#### - Inconvénients :

Il arrive qu'aucun releaser ne soit sélectionné. Dans ce cas, ce sont souvent les mêmes personnes qui prennent en charge le déploiement. Cela induit une charge de travail non équilibrée et une absence de partage de compétences qui elle-même induit la sélection du même releaser chaque semaine. C'est un cercle vicieux.

---

<sup>2</sup> Référent technique

L'équipe n'est pas forcément au fait de qui s'occupera du déploiement de la semaine, ce qui peut engendrer des quiproquos.

Il arrive qu'aucun développeur ne soit disponible pour le support. De plus, l'équipe de support niveau 2 envoie les requêtes via Skype aux mêmes développeurs (tech lead essentiellement).

Il n'y a pas de support fonctionnel attribué, ce qui peut générer des confusions dans les traitements (demandes envoyées et analysées par les développeurs au lieu d'être envoyées à un business analyst).

### Réconciliation GitHub/Jira

La release note est effectuée manuellement à chaque livraison par une réconciliation manuelle entre les tickets Jira et les commit GitHub.

#### - Inconvénients :

La rédaction de la release note prend du temps en raison du nombre de tickets Jira qui ne font pas l'objet de développement (support, clean code, ...). Certains tickets peuvent être omis de la release note, d'autres ajoutés à tort.

### *Sur le marché (Benchmark)*

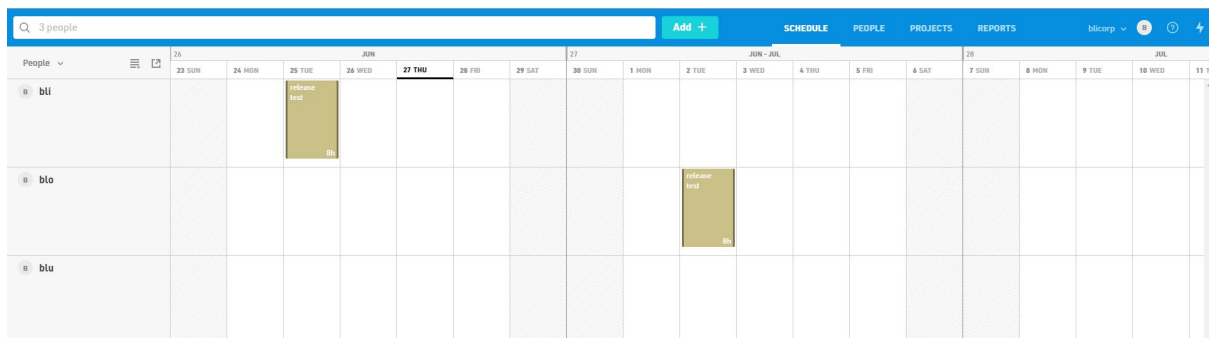
L'application devra répondre à tous les besoins tout en étant adaptable, évolutive et en respectant les contraintes de sécurité de l'entreprise.

### Gestion d'équipe

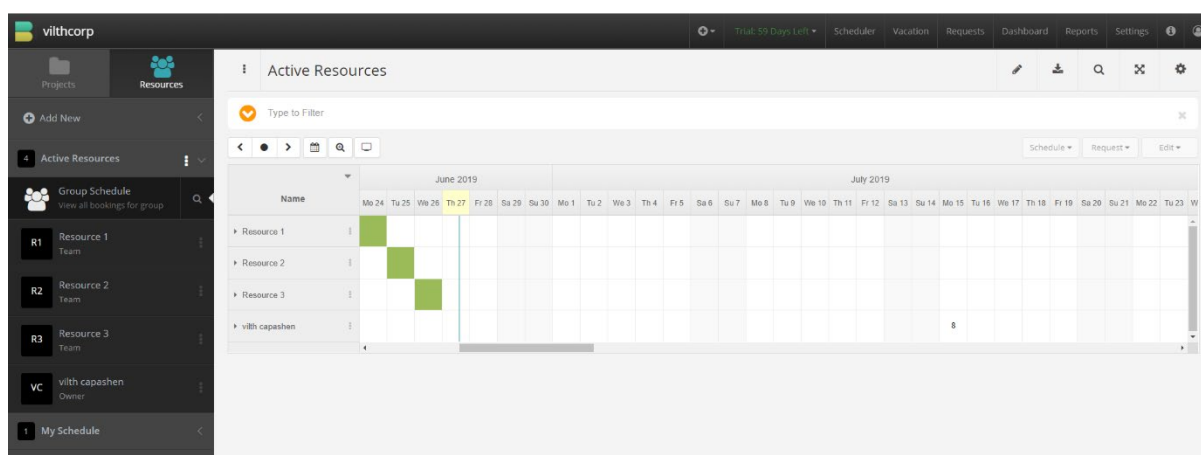
Un benchmark a été réalisé pour trouver des solutions de gestion d'équipe.

Les deux applications les plus proches du besoin sont Float et hubplanner, deux outils de gestion de tâches au sein d'une équipe.

### Float



## HubPlanner



Dans les deux cas, l'application proposée consiste en un calendrier dans lequel il est possible d'enregistrer des membres d'équipe et de mettre en place des tâches répétitives.

Ces outils ne répondent pas réellement au besoin de gestion d'équipe, en ce sens que bien qu'ils proposent une planification, la sélection de l'ordre de passage se fait manuellement en amont, et ne respecte pas le besoin de round robin. En outre, ces outils sont payants, ce qui est un frein évident.

### Réconciliation GitHub/Jira

Concernant le besoin de réconciliation entre GitHub / Jira, l'aspect sensible des données et les contraintes de sécurité (authentification sur les serveurs internes de ces outils) m'obligent à partir sur une solution interne.

### Solution préconisée

Devant l'absence d'offre pour la partie gestion d'équipe et l'obligation d'une solution interne pour la partie réconciliation, la solution préconisée doit être un développement interne.

Après analyse de l'existant en entreprise et de ce qui est proposé sur le marché il apparaît évident qu'il faudra développer une application en interne. Cela permettra de répondre aux impératifs de sécurité imposés par l'entreprise (système d'authentification OAuth2 d'entreprise, sensibilité des données notamment lors de l'accès aux serveurs d'entreprise GitHub et Jira) et de centraliser tous les besoins en une application.

Le premier livrable sera la sélection du membre en charge du déploiement. Cette fonctionnalité est déjà en partie implémentée dans le cadre du proof of concept, mais ne répond pas entièrement au besoin. Elle sera donc adaptée pour répondre aux besoins évoqués par les utilisateurs, à savoir l'automatisation / planification de la sélection, la modification d'un membre sélectionné au besoin.

La partie aide au déploiement est un besoin secondaire et interviendra dans un second temps, une fois partie gestion d'équipe fonctionnelle.

# GESTION DE PROJET

## Planning et suivi

### *Organisation du projet*

Le projet est organisé en utilisant la méthode agile.

Chaque itération est ponctuée par le déploiement du travail effectué et par une démonstration.

Cela permet un meilleur suivi de l'évolution du développement par les futurs utilisateurs et un retour régulier de ces derniers. Les adaptations éventuelles du besoin seront ainsi plus faciles et plus rapides, pour un produit final qui sera au plus près du besoin.

### User Stories

Les besoins exprimés par l'équipe sont traduits en tickets épiques et découpés en User Stories, afin de réduire la taille et donc la durée des tâches et pouvoir livrer de façon plus fréquente.

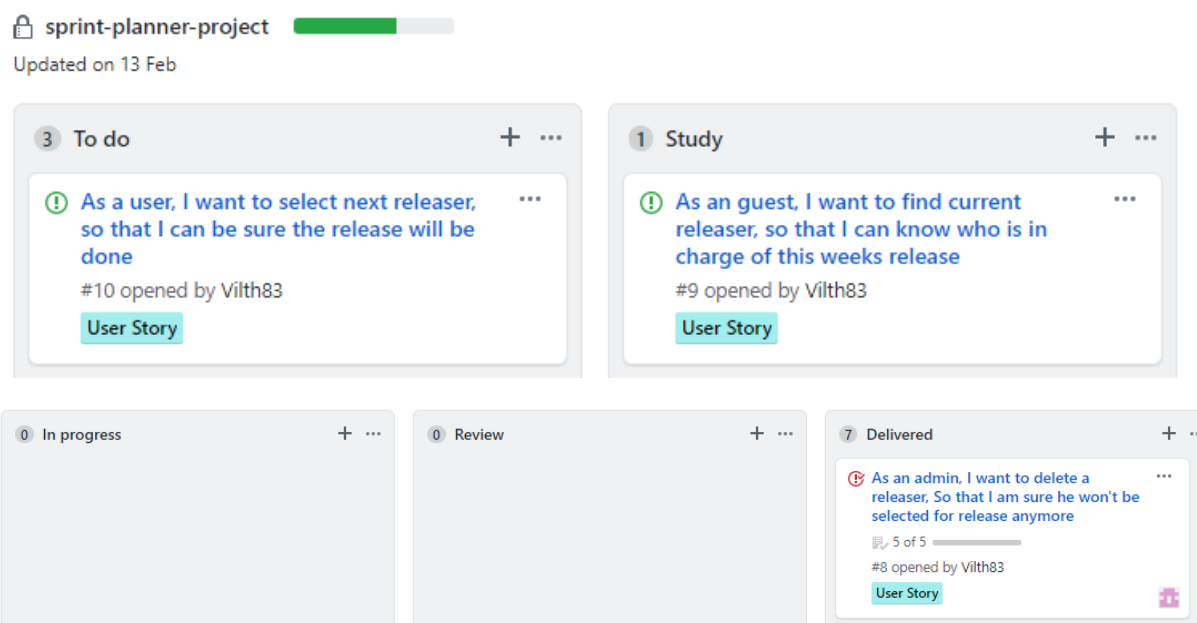
Les user stories servent de base pour les spécifications fonctionnelles. Elles devront donc être rédigées avec le plus grand soin et seront soumises à validation de l'équipe.

Elles devront contenir les éléments suivants :

- Le besoin exprimé au format « As a ... I want ... So that ... »
- Les spécifications fonctionnelles
- Les contraintes éventuelles
- Les critères d'acceptation

### Tableau de bord

Le projet est organisé à l'aide d'un tableau de bord Scrum. Les user stories sont organisées dans le backlog selon leur priorité dans le respect du besoin client et seront traitées dans l'ordre pour assurer des livraisons cohérentes et toujours utiles.



## *Planning*

En l'absence de date limite pour le projet et étant donné son organisation en mode agile, il n'y a pas lieu de mettre en place de planning.

Les itérations ont une durée de 3 semaines. Ce choix a été fait eu égard à l'agenda d'alternance (3 semaines d'entreprises pour 1 semaine de formation).

## *Suivi*

L'avantage principal de l'organisation agile est la mise en place des cérémonies.

### Daily meeting

Un suivi quotidien a lieu dans le cadre des Daily meetings. En effet, ma participation à cette cérémonie agile permet d'informer l'équipe de l'état d'avancement du projet et d'alerter en cas de point bloquant, afin d'obtenir l'aide d'un développeur de l'équipe.

### Démonstration client

Le suivi du développement de l'application est effectué à chaque fin d'itération sous la forme d'une démonstration aux clients. Ceux-ci peuvent ainsi voir l'état d'avancement du projet et soumettre leurs idées et réserves le cas échéant.

Ce format permet une adaptation constante au besoin du client tout en s'assurant à chaque itération que le besoin est bien couvert.

## Environnement humain et technique

### *Ressources humaines*

Je suis le seul développeur sur le projet.

Ma disponibilité fut variable pendant le développement. D'abord disponible pour le projet à temps plein, j'ai rejoint l'équipe sur leur application après 6 mois et n'ai alors été disponible que 2 jours par semaine.

Durant tout le développement, une semaine sur 4 fut dédiée à la formation en alternance.

### *Contexte technique*

#### Outils

Les utilisateurs disposent d'un PC et d'une connexion au réseau intranet. L'application n'est pas prévue pour être disponible en dehors du réseau de l'entreprise.

Elle sera hébergée sur un environnement interne et utilisée à partir d'un navigateur Web. Le navigateur de prédilection est Chrome. IE et Firefox ne sont pas des cibles prioritaires.

Les langages utilisés ne sont pas imposés. Cependant, il conviendra de prendre en considération les langages maîtrisés par l'équipe dans le choix des langages, et certaines contraintes concernant les versions et la base de données seront à prendre en compte.

## Contraintes

Les contraintes sont les suivantes :

- La base de données sera une base embarquée, qui devra être persistante. En effet, il ne sera fourni qu'un environnement restreint sans installation de serveur de base de données pour des raisons de coût.
- L'application étant hébergée en interne et disponible uniquement sur l'intranet derrière le proxy d'entreprise, l'authentification n'est pas une priorité.
- La charte graphique de l'entreprise devra impérativement être respectée.
- Les dépendances et librairies utilisées devront être obtenues à partir de sources internes. Les versions plus récentes mais non disponibles en interne ne devront pas être utilisées.

## Objectifs de qualité

### Disponibilité

L'application devra être disponible pour les utilisateurs sur les horaires de travail de France et d'Inde. Il n'est pas exclu qu'une équipe nous rejoigne au Canada (besoin de support en « Follow the Sun »). Dans ce cas, l'application devra être disponible 24h / 24h.

De plus, l'automatisation des tâches de l'application implique que le serveur soit disponible au moment des événements.

### Facilité d'utilisation

L'application est destinée à une trentaine d'utilisateurs venant de deux pays différents et représentant deux métiers différents. Bien qu'ils soient tous relativement à l'aise avec l'outil informatique, l'application devra être ergonomique afin qu'ils aient rapidement et facilement les informations recherchées.

Toute l'équipe communiquant en grande majorité en anglais, la langue anglaise devra être utilisée. Il ne sera pas utile de mettre en place une traduction de la page mais cela pourrait être un plus à long terme.

### Sécurité

Comme indiqué précédemment, l'authentification n'est pas une priorité. Cependant, l'application sera amenée à faire appel aux API Jira et GitHub internes. Jira nécessitant une authentification, il conviendra de la mettre en place en sécurisant les identifiants utilisés par l'utilisateur. A ce titre, aucun mot de passe ne devra être stocké et il devra être envoyé à Jira crypté.

Le code de l'application est propriété de l'entreprise et ne devra pas être mis en ligne.

### Tests

L'application devra respecter les standards de l'équipe. La couverture de code du serveur (back end) devra être supérieure ou égale à 70%.

Le code devra être testé par des tests unitaires et des tests d'intégration. L'application étant légère, il n'y a pas lieu d'effectuer des tests de performance.

Les tests UAT seront réalisés à l'aide de scénarios prédéfinis par les utilisateurs en adéquation avec l'expression de besoins.

# SPECIFICATIONS FONCTIONNELLES ET TECHNIQUES



# SPECIFICATIONS FONCTIONNELLES

## Contexte

L'équipe est séparée en deux Feature teams. Chaque Feature Team a son propre Agile Master.

Tous les développeurs travaillent sur un projet d'application à usage interne de gestion de risque en Agile@Scale. Les itérations sont les suivantes :

- 10 semaines de P.I. correspondant à 5 sprints.
- Des sprints de 2 semaines
- Des déploiements chaque semaine (2 déploiements par sprint)

## Acteurs

### *Agile Master*

#### Description

Les agile masters ont le rôle d'administrateur, afin de pouvoir gérer les membres, les candidats au déploiement, support, tests et de modifier les données liées au projet. Ils gèrent également la validation des comptes en attente. Ils ont des droits de création, modification et suppression sur toutes les ressources.

#### Persona

## Bert Vaner



age : 39

Job : Business Analyst

Role: Admin

Status : contractor

*I spend a lot of time trying to organize team without any informations concerning release. I need a tool to manage that for me !*

#### Biography

Bert is in charge of functional needs. He also organize iterations, morning meetings and sprint debriefing.

He has worked for the project for 2 years. Previously in a team of 2 BAs, he is now alone as his coworker quitted his job. Bert as taken a new task of agile master when his coworker leaved

Each week, he looks in JIRA and GITHUB for finished features and send a mail to the team. He can also be a support team leader.

#### Goals

- Manage sprint iterations
- Ease his own work

#### Frustrations

- Have difficulties to retrieve informations about the current sprint
- Have to do a lot of work manually


## Développeurs / Business Analysts

### Description

Les membres de l'équipe sont les utilisateurs principaux. Ils ont le rôle d'utilisateur. Ils peuvent accéder aux informations, modifier le candidat à un déploiement ou au support et accéder à l'outil de réconciliation GitHub/Jira.

### Persona

## Jay Devoe



**Biography**

Jay is a senior software engineer working in the society for many years. He acts like a fireman on urging topics, and doesn't code that much anymore.

He is the one the team relies on when a problem occurs, or when difficulties are encountered regarding release.

**age** : 48

**Job** : Senior software engineer

**Role** : user

**Status** : Employee

*I want to have a precise insight of the release process : who is in charge, which step are we in, is everything fine or not?*

**Goals**

- Develop features
- Know who take cares of the release and the support team
- Spend less time debbuging team members work

**Frustrations**

- Is always bothered by other problems than his own features
- Is called at the last moment to release or support when anyone else has been selected
- Never know who is releaser, support or if noone is in charge

## Autres utilisateurs

### Description

Les invités peuvent accéder à la page d'accueil afin d'avoir la synthèse des informations (membres désignés au déploiement/ support) et peuvent créer un compte. Le compte devra être validé par un administrateur.

### Persona

## Terry Valliantown



**Biography**

Terry is in an internship in the company. He is here to learn development and project management.

As a new developer, he have got a lot to learn, and have difficulties to get help and informations about the project and the process. Other developers in the team are indeed overwhelmed by their own work and don't have much time to share with Terry.

He wants to join the support team to improve his skills.

**age** : 35

**Job** : Junior software engineer

**Role** : guest

**Status** : Intern

*I want to have a precise insight of the release process : who is in charge, which step are we in, is everything fine or not?*

**Goals**

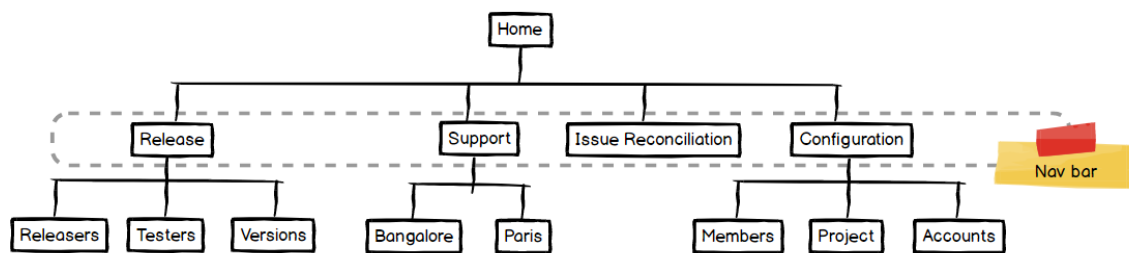
- Learn the job
- Be useful to his team

**Frustrations**

- Is left alone as teammates don't have time to spend with him
- Does not know who he can bother and who he can't
- Want to be an help to the team rather than a problem
- Does not know how release process works

## Fonctionnalités principales

### Sitemap générale



### Page d'accueil

#### Cas d'utilisation

Un utilisateur souhaite savoir qui est désigné pour les différentes tâches à traiter. L'information est disponible que l'utilisateur soit identifié ou non, dès la page d'accueil.

#### User Story

As a guest,  
I want to find current candidate,  
So that I can contact him if needed

#### specs

- ☐ Display the candidate with his task
- ☐ separated display for support and release related informations
- ☐ release number concerned by this selection

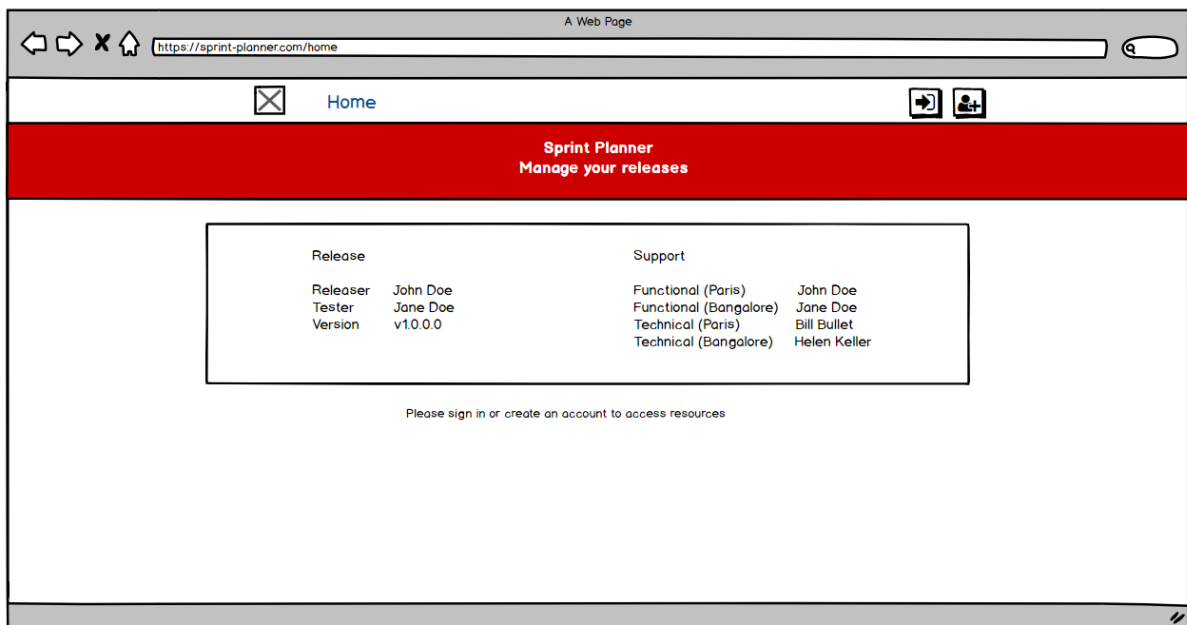
#### Acceptance Criteria

- ☐ Every candidate is displayed if given
- ☐ Display should be done for guest, logged in user and logged in admin

### Sitemap



## Wireframes



### Tâche automatisée

#### Cas d'utilisation

Il s'agit des tâches effectuées automatiquement par l'application. Aucune action humaine n'est requise. L'exécution est prévue pour le lundi à 02 : 00 AM CET (Central European Time soit heure de Paris) afin que l'équipe soit informée du nom des candidats sélectionnés dès son arrivée au bureau, que ce soit à Paris ou Bangalore.

#### Prérequis

Pour qu'un candidat soit sélectionné comme releaser, il faut :

- Qu'au moins un membre soit enregistré dans l'application
- Qu'au moins un membre soit défini comme candidat à la tâche dans l'application

#### User Story

As a releaser candidate  
I want to be picked up automatically  
So that I can take care of the release

##### specs

- ☐ CRON trigger to execute the scheduled task

##### Acceptance Criteria

a releaser should be picked up with round robin logic :

- ☐ All priorities should be incremented by 1
- ☐ current Releaser should become available and its priority should be set to 0
- ☐ all unavailable candidates should become available
- ☐ candidate with the highest priority should be designated for release
- ☐ an email should be sent to the designated releaser and to the team

L'exécution étant automatique, cette fonctionnalité n'est pas concernée par le sitemap et les wireframes.

### *Gestion des candidats*

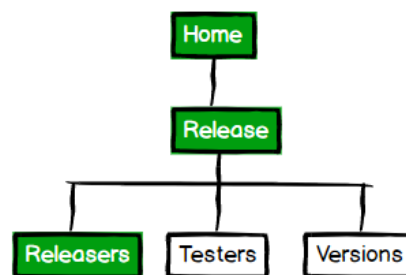
La gestion des candidats regroupe les fonctionnalités de sélection automatique, sélection manuelle, visualisation, ajout et suppression des candidats à des tâches.

Comme indiqué précédemment, les tâches sont aujourd'hui les suivantes : Releaser, Support et Tester. La réalisation des fonctionnalités doit permettre de gérer et d'afficher les candidats aux différentes tâches.

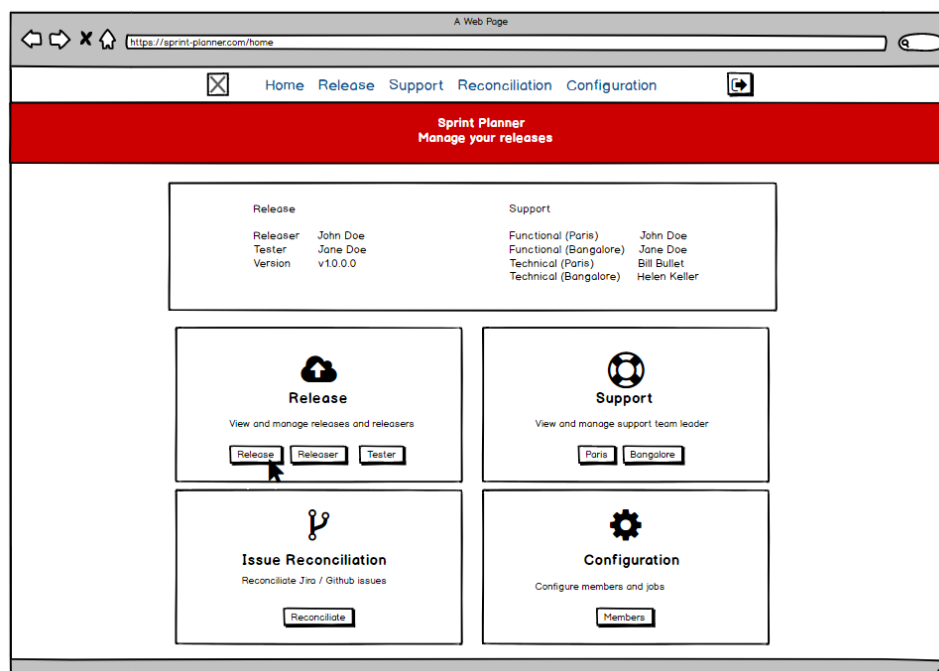
### Prérequis

L'accès aux ressources de gestion des candidats nécessite une authentification avec des droits d'utilisateur ou d'administrateur et partagent les éléments suivants.

### Sitemap



### Wireframes



## Sélection du prochain candidat prévu

### Cas d'utilisation

Un releaser a été désigné par l'application mais il ne pourra pas assurer le suivi des développements et / ou le déploiement car il est en congés. Il faut donc désigner le prochain candidat prévu pour le remplacer.

### User Story

As a logged in user,  
I want to be able to select next releaser  
So that the release can be done if current releaser is unavailable

#### specs

☐ Status : UNAVAILABLE, AVAILABLE, CURRENT

#### Acceptance Criteria

- ☐ A user should be able to change to next releaser
- ☐ A CURRENT releaser should become unavailable when functionality is used
- ☐ the highest priority AVAILABLE candidate should become CURRENT releaser
- ☐ If no releaser is found, an error message should be thrown, to inform that no releaser has been found.

### Wireframes

#### Vue depuis un compte utilisateur

The wireframe shows a web browser window with the URL <https://sprint-planner.com/home>. The page has a navigation bar with links: Home, Release, Support, Reconciliation, and Configuration. Below the navigation bar is a red header section titled "Releaser Releaser Management". The main content area contains three sections: "Current releaser" (a grey box), "Releaser" (a white box showing "John Doe" with a swap icon), and "Releaser candidates" (a grey box). Below the "Releaser candidates" section is a table with the following data:

Priority	Firstname	Lastname	Status
1	Jane	Doe	AVAILABLE
2	Bill	Bullet	UNAVAILABLE
3	Helen	Keller	AVAILABLE
4	John	Doe	CURRENT

At the bottom of the page, there is a section titled "Add a new releaser" with a dropdown menu showing "Sarah Connor" and a save icon.

## Sélection manuelle d'un candidat

### Cas d'utilisation

Un releaser a été désigné par l'application. Cependant, la criticité du prochain déploiement nécessite qu'il soit réalisé par un référent technique. Il faut donc désigner le prochain candidat manuellement.

### User Story

As a logged in user,  
I want to be able to change candidates status  
So that I can ensure a specific member will oversee the release

#### specs

☐ Status : UNAVAILABLE, AVAILABLE, CURRENT

#### Acceptance Criteria

- ☐ A user should be able to modify status of a candidate
- ☐ If a candidate is set as CURRENT, the former CURRENT should become unavailable

### Wireframe

#### Vue depuis un compte administrateur

The wireframe shows a web browser window with the URL <https://sprint-planner.com/home>. The page has a navigation bar with links: Home, Release, Support, Reconciliation, and Configuration. Below the navigation bar is a red header section titled "Releaser" and "Releaser Management".

The main content area is divided into two sections:

- Current releaser:** A box containing the text "Releaser" and "John Doe" with a refresh icon.
- Releaser candidates:** A box containing a table of candidates and an "Add a new releaser" button.

Priority	Firstname	Lastname	Status	Delete
1	Jane	Doe	CURRENT	
2	Bill	Bullet	AVAILABLE	
3	Helen	Keller	UNAVAILABLE	
4	John	Doe	CURRENT	
			AVAILABLE	

Below the table, there is an "Add a new releaser" button and a dropdown menu with the name "Sarah Connor" and a save icon.

## Réconciliation GitHub / Jira

### Cas d'utilisation

Le déploiement est imminent. Le business analyst doit écrire la note de déploiement qui reprendra les fonctionnalités déployées afin de la transmettre au client pour validation. Il doit donc trier parmi tous les tickets Jira ceux qui correspondent aux nouvelles fonctionnalités, aux corrections de bug en excluant les tickets de support, les tickets techniques et les tickets d'analyse.

### User Story

**As a user,**

**I want to reconcile github and jira**

**So that I can easily retrieve the content of the release**

*specs*

- ☐ Current version, mandatory (candidate to release)
- ☐ Previous version, mandatory (version to compare with)
- ☐ Github repository, optional

*workflow*

- ☐ Retrieve Github concerned commits
- ☐ Retrieve Jira associated issues
- ☐ Reconciliate Github and JiraIssues

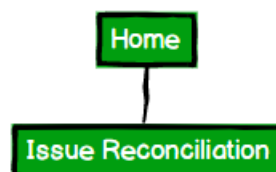
*constraints*

- ☐ Handle dependency to external api and errors that might happen

*acceptance criteria*

- ☐ Should return a list of reconciliated issues
- ☐ should return an error if missing current or previous branch
- ☐ should reconcile default repository if not specified
- ☐ should reconcile given repository if specified
- ☐ should aggregate Jira and GitHub informations
- ☐ Should warn in case of unreachable external API (GitHub or Jira)

### Sitemap





## Wireframe

A Web Page

https://sprint-planner.com/home

Home Release Support Reconciliation Configuration

### Issue Reconciliation Reconciliation between Github And Jira

Branch selection

Select branches for comparison

Current branch \* v1.0.0 Previous branch \* v1.0.0 repository (optional)

Retrieve reconciliated issues

Reconciliated issues

Ticket	Proof	State	Merged	Title	Fix version
SPL-1	v	v	25-02-2019	[SPL-1] Initiate spring project	v1.0.0
SPL-2	v	v	27-02-2019	[SPL-2] Initiate Angular project	v1.0.0
SPL-3	x	v	28-02-2019	[SPL-3] Configure Jenkins and Sonar	
SPL-4	v	v	01-03-2019	[SPL-4] declare ORM bean	v1.0.0
SPL-5	v	x	01-02-2019	[SPL-5][CLEANCODE] Fix sonar issues	v1.0.0

## SPECIFICATIONS TECHNIQUES

### Ecosystème de données (stack technique)

Le choix de la stack technique est soumis à des contraintes de l'entreprise : Les dépendances requises sont disponibles en interne via l'outil Nexus. Cet outils fourni les dépendances après qu'elles ont été validées par le service d'architecture. Les dernières versions des librairies ne sont donc pas forcément disponibles.

La sélection des outils a été faite après analyse des solutions existantes, en veillant à rendre l'application portable, maintenable et évolutive. Au vu du faible trafic et du volume de données, la scalabilité n'est pas une priorité.

L'écosystème préexistant dans l'équipe a été un élément de décision crucial. En effet, le fait d'utiliser des outils avec lesquels toute l'équipe est à l'aise permet de faciliter le support et la maintenabilité.

#### Serveur

L'application sera effectuée sous forme d'API<sup>3</sup> REST<sup>4</sup> en Java 8 à l'aide des outils Spring 2.7 et Maven 3.6.

Java permet que l'application soit portable sur n'importe quel système équipé d'un JRE, tandis que Spring fournit de nombreux avantages en termes de développement d'API. Maven facilitera le déploiement de l'application et la gestion des librairies nécessaires au développement.

#### Client

Le client sera développé en Angular 7. Angular est un framework complet, mature, avec une documentation étoffée. De plus, l'utilisation de TypeScript permet un typage strict des variables et a des similitudes avec JAVA, ce qui permet de faciliter le support.

Les librairies utilisées seront notamment AgGrid pour les tableaux et NgxBootstrap pour la mise en page.

#### Base de données

La principale contrainte apportée par l'équipe est l'absence de serveur de base de données pour la réalisation du projet. J'ai donc opté pour une base embarquée H2.

Cette dernière permet à la fois d'avoir une base persistée sous forme de fichier et de bénéficier au besoin d'une base dite « in-memory », c'est-à-dire non persistante, qui facilitera les jeux de test d'intégration.

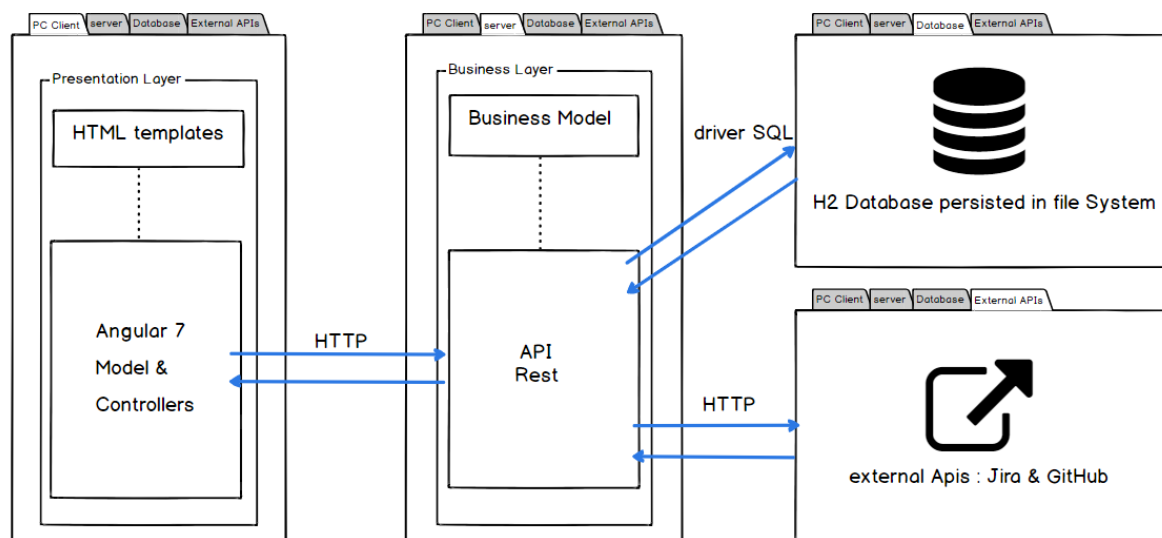
---

<sup>3</sup> Application Programming Interface

<sup>4</sup> REpresentational State Transfer

## Base de données et diagrammes

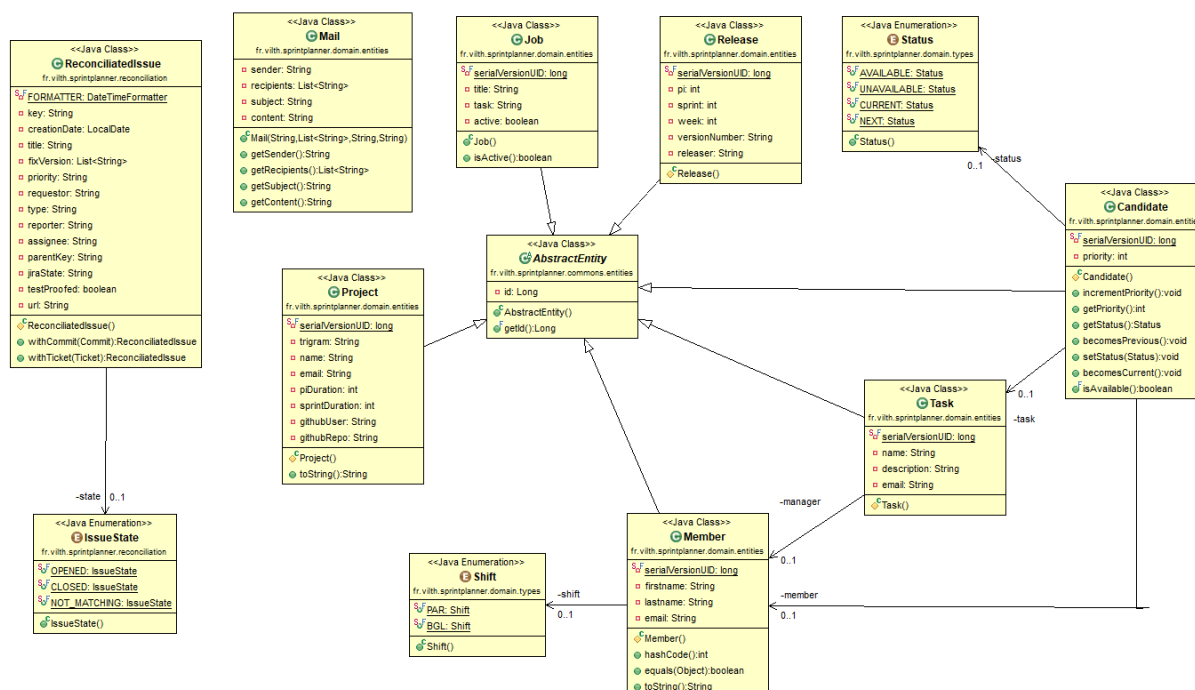
### Diagramme d'architecture



### Diagrammes de classes

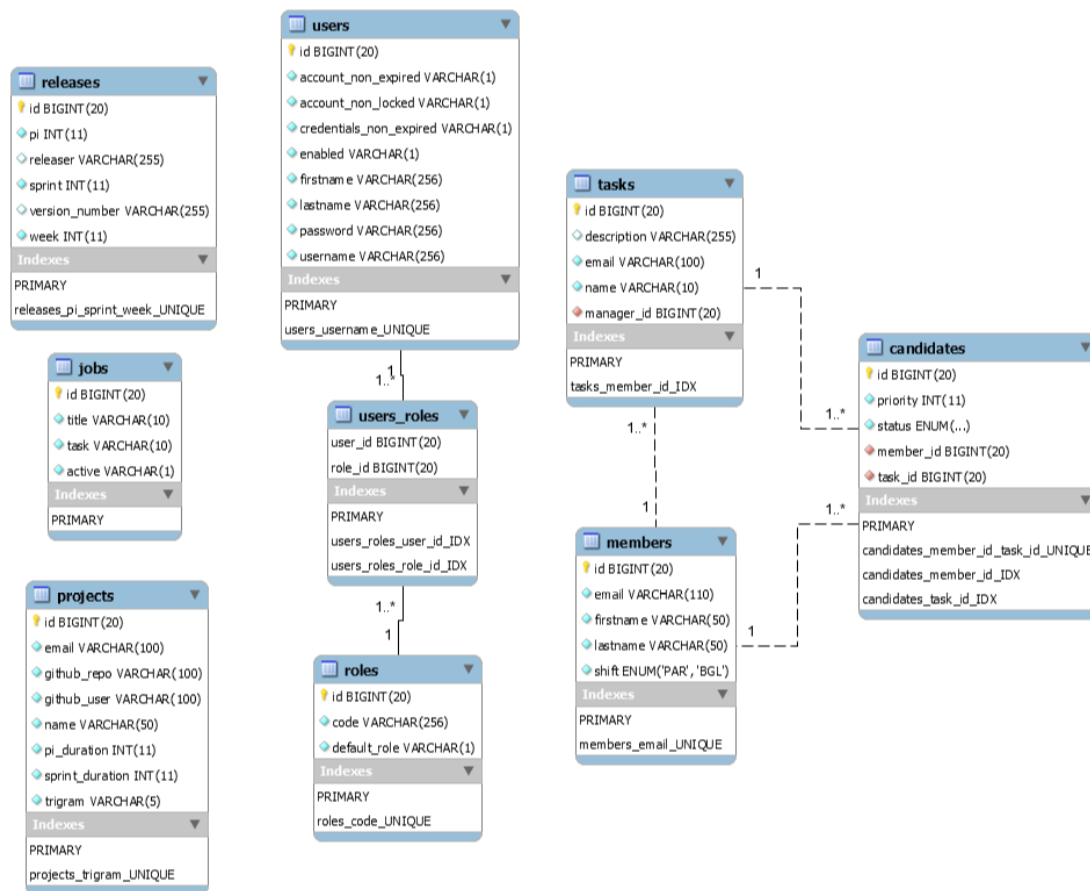
Le diagramme ci-dessous représente les classes du domaine de l'application.

Les autres diagrammes de classe correspondant aux connecteurs Jira / GitHub et à l'authentification notamment sont disponibles en annexe 1.



## Base de données

Le diagramme ci-dessous représente le modèle physique de données. Il a été édité sur MySQL Workbench.



## Règles d'implémentation

### Grands principes

#### Couches

Le diagramme d'architecture correspond à une architecture n-tiers : la partie client (couche client), la partie server (server TOMCAT embarqué dans une application SpringBoot) (couche applicative) et la partie base données (couche de persistance) (Base embarquée H2 persistée sur fichier). Les deux API externe GitHub et Jira sont également représentées (couche api externes).

#### Mapping ORM<sup>5</sup>

Le mapping des entités vers la base de données est traité par Hibernate/JPA<sup>6</sup> qui permet de faire le mapping objet-relationnel par annotation.

<sup>5</sup> Object Relational Mapping

<sup>6</sup> Java Persistence API

Cela permet de contrôler le nommage des colonnes, les contraintes et ainsi d'avoir une meilleure maîtrise de la base de données. De plus le mapping permet d'ajouter les contraintes de la base dans les entités de l'API et donc de protéger la base de données contre des données non attendues.

Il est à noter que la mise en place des contraintes en base par JPA permet d'avoir la connaissance parfaite des contraintes en base, ce qui facilite le choix des annotations de validation dans les DTO.

## *Sécurité*

### Authentification

L'authentification est gérée par JWT<sup>7</sup> afin d'assurer une authentification Oauth2. L'utilisation de JWT permet de limiter le nombre de requêtes, puisque le jeton contient les informations nécessaires à l'identification de l'utilisateur et contient ses autorisations.

Il sera stocké par le client dans un cookie ayant une expiration d'une journée. Ce délai a été choisi en cohérence avec les changements d'équipe : un membre d'équipe qui change d'affectation prend en général ses nouvelles fonctions le lundi, et quitte donc notre équipe le vendredi. La désactivation de son compte sur Sprint-planner permet qu'il n'ait plus accès aux ressources à son retour de week-end.

### Autorisations

Deux rôles ont été identifiés : utilisateur (USER) et administrateur (ADMIN). Il n'est pas exclu que d'autres rôles soient ajoutés.

L'application des rôles sera implémentée par annotation au niveau back end grâce à la librairie Spring Security. Un intercepteur http dans la couche client de l'application permettra d'envoyer le jeton à chaque requête pour l'envoyer à la couche serveur.

## *Tests*

### Unitaires

Les tests unitaires serviront à s'assurer du fonctionnement des méthodes d'entités, de DTO et des méthodes contenant la logique métier. Les méthodes impliquant des dépendances avec d'autres composants feront l'objet de tests d'intégration.

Les données de test seront injectées à partir de fichiers CSV pour faciliter la lisibilité des tests et la modification des données de test.

### D'intégration

L'intégration des composants de bout en bout devront être testés à l'aide de tests d'intégration. Un DML<sup>8</sup> sera mis à disposition afin de centraliser les données et de peupler la base de tests au moment de l'exécution.

Un profil de déploiement « test » sera créé afin d'effectuer les tests avec la configuration adéquate. Le fichier de propriété est disponible sur le dépôt Github et les propriétés seront injectées par Jenkins, notamment concernant les mots de passe de la base de données.

---

<sup>7</sup> Json Web Token

<sup>8</sup> Data Manipulation Langage : script en charge notamment de l'insertion de données

## Tests utilisateurs

Des tests UAT<sup>9</sup> seront rédigés et soumis à des utilisateurs. Cela permettra qu'ils s'exercent à l'utilisation de l'application, fournissant ainsi un retour sur la navigabilité tout en testant les scénarios afin de s'assurer de la cohérence de la réponse au besoin fonctionnel.

## Non-régression

L'exécution des tests unitaires, des tests d'intégration et des tests UAT avant chaque mise en production est à considérer comme la batterie de test de non-régression.

Dès lors qu'une régression est détectée, la mise en production est annulée et l'anomalie se doit d'être corrigée.

Une exception pourra être tolérée dans le cadre d'échec de tests UAT, à la condition expresse que le client statue sur la gravité de cette régression et accepte un déploiement avec cette erreur. Elle devra alors être corrigée dans les plus brefs délais par le déploiement d'un fix.

## Déploiement

### Jenkins & Sonar

L'outil Jenkins sera utilisé pour le contrôle de l'application. Chaque nouveau déploiement devra donner lieu à une compilation par Jenkins qui effectuera un build et lancera les jeux de tests pour analyse par Sonar.

### XL-Deploy

Le déploiement sera assuré par l'outil XL-Deploy afin de faciliter et accélérer le processus. Des scripts de démarrage et d'arrêt de l'application devront donc être rédigés afin de permettre un déploiement de « bout en bout ».

### Déploiement tout-en-un

Afin de faciliter le déploiement de l'application, les ressources client sont encapsulées dans les ressources serveur. Ainsi, un simple démarrage du serveur permet d'accéder à un IHM sans autre déploiement. La couche client est cependant déployable en parallèle de la couche serveur au besoin.

## Bonnes pratiques

### Conventions de nommage

Pour assurer la cohérence dans le code, l'implémentation devra suivre des règles de nommage strictes. La couche serveur respectera les conventions Javabeau<sup>10</sup>. La couche de présentation suivra ces conventions afin d'assurer une cohérence avec le back-end. La base de données suivra les conventions de nommage SQL, notamment le « snake\_case ».

---

<sup>9</sup> User Acceptance Tests

<sup>10</sup> Conventions de nommage en vigueur pour Java, indispensables au bon fonctionnement de certaines librairies, notamment JPA et Jackson

## *Programmation défensive*

### Instanciation d'objets

La grande majorité des objets ont un constructeur protégé sans argument, afin de limiter l'instanciation d'objet dans le code. C'est notamment le cas des entités. Cela permet de contrôler les modifications de données dans le code, en laissant la responsabilité de créer les objets au Framework Spring.

### Gestion des erreurs

Dans le cadre de la programmation défensive, les données en entrée seront validées par le client (validation de saisie) et par le serveur (validation des DTO par annotations).

Un ControllerAdvice gèrera les exceptions rencontrées à l'exécution. Celui-ci a pour rôle d'intercepter les erreurs de tous les contrôleurs de l'application et de leur apporter le traitement nécessaire afin qu'ils soient restitués de manière claire à l'utilisateur.

Les erreurs non gérées (erreur 500) devront être loguées avec leur cause afin qu'un suivi soit assuré et le code amélioré, dans une démarche d'amélioration continue.

### Qualité de code

Afin d'assurer la maintenabilité du code, la qualité du code sera revue à chaque nouveau développement. Une analyse du code avec Jenkins sera faite à chaque fois que du code est proposé et le code sera soumis à l'outil SonarQube pour vérifier l'absence de bug, failles de sécurité et vérifier la couverture du code. Le projet doit avoir une note globale égale à A et une couverture de code d'au moins 70%.

## *Design patterns*

Les design pattern suivants seront utilisés lors du développement de l'application :

### Facade

Les déclencheurs des jobs automatisés sont développés avec le design pattern façade : La méthode utilisée pour exécuter les jobs est dans la classe JobTrigger. Elle-même appelle des méthodes réparties dans des classes avec des responsabilités bien définies : MailJob pour l'envoi de mails, ReleaseJob pour la gestion des releasers et de la release et SupportJob pour la gestion du support.

### DTO

Omniprésent dans le projet, le design pattern DTO permet de limiter les appels à la base de données en regroupant les éléments dans un « Data Transfer Object ». Cela permet d'encapsuler toutes les informations requises en un objet.

Il a en outre l'avantage de permettre de restreindre l'accès aux entités métier que ce soit en lecture et en écriture en ne donnant accès qu'à des représentations de ces entités. Ce Design pattern permet de faciliter la validation des entrées utilisateur et de contrôler rigoureusement ce qui entre et sort de l'application.

### Singleton

Spring utilise le design pattern Singleton pour instancier les « bean », objets créés au démarrage de l'application. Cela permet d'appliquer l'injection de dépendances et facilite la mise en place de l'API.

Ainsi, tous les contrôleurs, services, repositories et plusieurs objets utilitaires (ObjectMapper, ModelMapper, ...) sont des singletons.

Angular déclare tous les services « injectables » (annotés @injectable) comme des singletons.

Dans le cadre du développement, les logs mis en place pour la maintenance et le contrôle des modifications sont effectués via un logger sous forme de singleton.

### Front Controller

Le design pattern front controller est utilisé par Spring pour la gestion des requêtes. Spring dispose d'un dispatcher qui redirige les requêtes vers les contrôleurs de l'application grâce à leur mapping.

Ainsi, les requêtes sont toutes interceptées au même endroit et la gestion des erreurs est facilitée par un controllerAdvice (cf. implémentation > couche serveur > API > contrôleurs).

### Javadoc

Le code JAVA est entièrement documenté sous forme de JAVADOC. Celle-ci est disponible dans le code lui-même et sous forme de fichiers HTML à la disposition des développeurs qui rejoindraient le projet.

### Les Six Commandements

L'application devra respecter six commandements, qui sont de bonnes pratiques assurant un code propre, compréhensible, maintenable et réutilisable : DRY<sup>11</sup>, YAGNI<sup>12</sup>, KISS<sup>13</sup>, ACID<sup>14</sup>, SOLID<sup>15</sup> et la loi de Déméter.

DRY : Eviter les répétitions de code favorise la réutilisabilité, la **maintenabilité** et l'**adaptabilité** du code.

YAGNI : N'implémenter que ce qui est nécessaire permet d'assurer la **simplicité** du code et d'éviter le code dit « mort ».

KISS : Simplifier le code permet de faciliter sa lisibilité et donc sa **maintenabilité** par d'autres développeurs ne connaissant pas le projet.

SOLID : ensemble de principes de conception facilitant la **flexibilité** et la **maintenabilité** d'un programme.

ACID : ensemble de propriétés **garantissant** les **transactions** en base de données.

Loi de Déméter : Un objet ne devra avoir une connaissance que de lui-même et de ses objets membres pour permettre une meilleure **maintenabilité** et **adaptabilité**.

---

<sup>11</sup> Dont Repeat Yourself

<sup>12</sup> You Ain't Gonna Need It

<sup>13</sup> Keep It Stupid Simple

<sup>14</sup> Atomicité, Cohérence, Isolation, Durabilité

<sup>15</sup> Single Responsibility Principle, Open/Close, Liskov Principle, Interface Segregation, Dependency Inversion

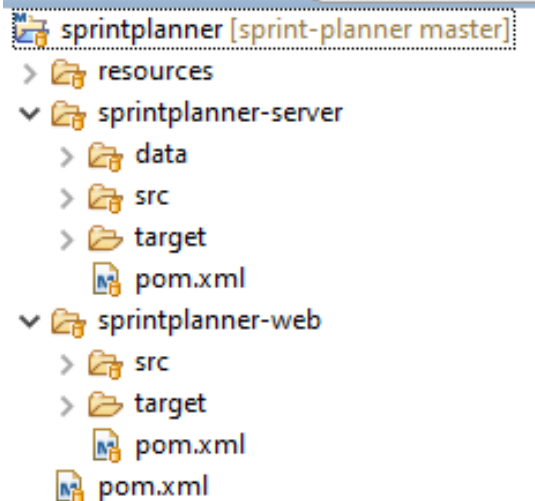


# IMPLEMENTATION ET TESTS

# IMPLEMENTATION

## Organisation du code

### Module parent



L'application est constituée d'un package parent, servant à compiler la totalité du code.

Cette compilation est effectuée à l'aide de Maven, via un POM<sup>16</sup> parent.

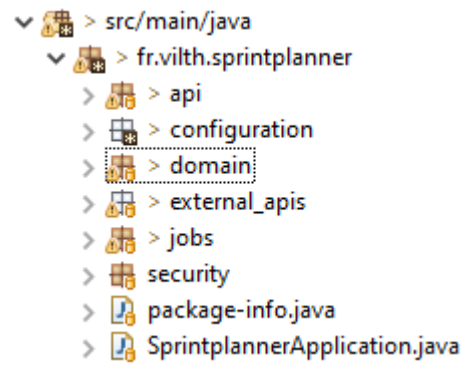
Le POM Parent déclare les modules server et client dont il exécute la compilation via leurs POM propres.

Les packages sont donc organisés suivant cette hiérarchie

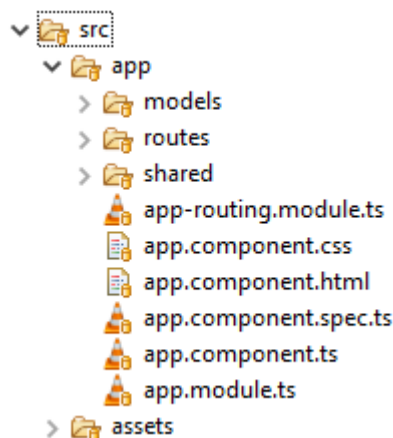
### Serveur

Les package du serveur ont été découpés pour faciliter la recherche dans le code :

- Composants d'accès à l'API (API)
- Entités et DTO (domain)
- Services d'accès aux API externes
- Tâches automatisées (jobs)
- Configuration des beans (configuration)
- Configuration propre à la sécurité



### Client



La partie client est découpés en trois packages :

Le package model contient les définitions d'objets (DTO).

Le package routes contient les controllers et vues de chaque route de l'application web.

Le package shared contient les composants partagés entre les routes (controleurs et vues communes, services, gestion de la sécurité, gestion des erreurs, ...).

<sup>16</sup> Project Object Model

## Base de données

### Définition du schéma

La rédaction des scripts de création et d'insertion de données en base a été édités à partir du workbench SQL. Afin de faciliter le passage d'une base de données H2 à une base de données MySQL, deux scripts de DDL<sup>17</sup> ont été édités et mis à disposition.

#### Base H2

```
--create schema
CREATE SCHEMA IF NOT EXISTS `sprintplanner` AUTHORIZATION root;
USE `sprintplanner`;

-----
-- Table `sprintplanner`.`members`
-----

CREATE TABLE IF NOT EXISTS `sprintplanner`.`members` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(110) NOT NULL,
  `firstname` VARCHAR(50) NOT NULL,
  `lastname` VARCHAR(50) NOT NULL,
  `shift` ENUM('PAR', 'BGL') NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE `members_email_UNIQUE` (`email` ASC));
```

L'exemple ci-dessus a été créé à des fins de test. Ainsi, l'utilisateur en base de données est 'root' et a les autorisations de modification de la base. En production, la base de données est fournie déjà préremplie des tables de configuration et n'est pas accessible aux utilisateurs. Seul le support (dans les faits, moi seul) est habilité à effectuer des modifications, et est seul détenteur des identifiants.

#### MySQL

```
-- Schema sprintplanner
CREATE SCHEMA IF NOT EXISTS `sprintplanner`
DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci;
USE `sprintplanner`;

-----
-- Table `sprintplanner`.`members`
-----

CREATE TABLE IF NOT EXISTS `sprintplanner`.`members` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(110) NOT NULL,
  `firstname` VARCHAR(50) NOT NULL,
  `lastname` VARCHAR(50) NOT NULL,
  `shift` ENUM('PAR', 'BGL') NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `members_email_UNIQUE` (`email` ASC) VISIBLE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

Le script de création de schéma MySQL est très similaire au script H2, à deux détails près :

- Le dialecte MySQL permet la gestion de l'encodage des caractères et de la collation

---

<sup>17</sup> Data Definition Language, script de définition du schéma

- Il est possible de déclarer des index. Cela accélère les requêtes en base.

#### Insertion de données

Les données nécessaires au fonctionnement de l'application (tables de configuration) sont créées au moyen d'un script d'insertion de données. S'agissant d'ajout de données, ils sont identiques que la base utilisée sur H2 ou MySQL.

```
insert into tasks (description, email, name, manager_id) values
('manage releases', 'releaser@mail', 'releaser', 1),
('manage support', 'support@mail', 'technical', 1),
('handle tests', 'test@mail', 'tester', 1),
('manage support', 'support@mail', 'functional', 1);
COMMIT;
```

#### Sélection du candidat par l'algorithme de round robin

##### *Argumentation*

L'algorithme principal utilisé est le round robin afin de déterminer le candidat à une tâche.

Le principe du round robin est le principe du tourniquet : les candidats sont dans une file d'attente pour gérer une tâche. Une fois un candidat sélectionné, il effectue sa tâche pour une durée donnée et rejoint le début de la file d'attente. Si un candidat ne peut effectuer sa tâche à son tour il cède sa place au candidat suivant et reste alors au début de la file afin d'être le prochain sélectionné.

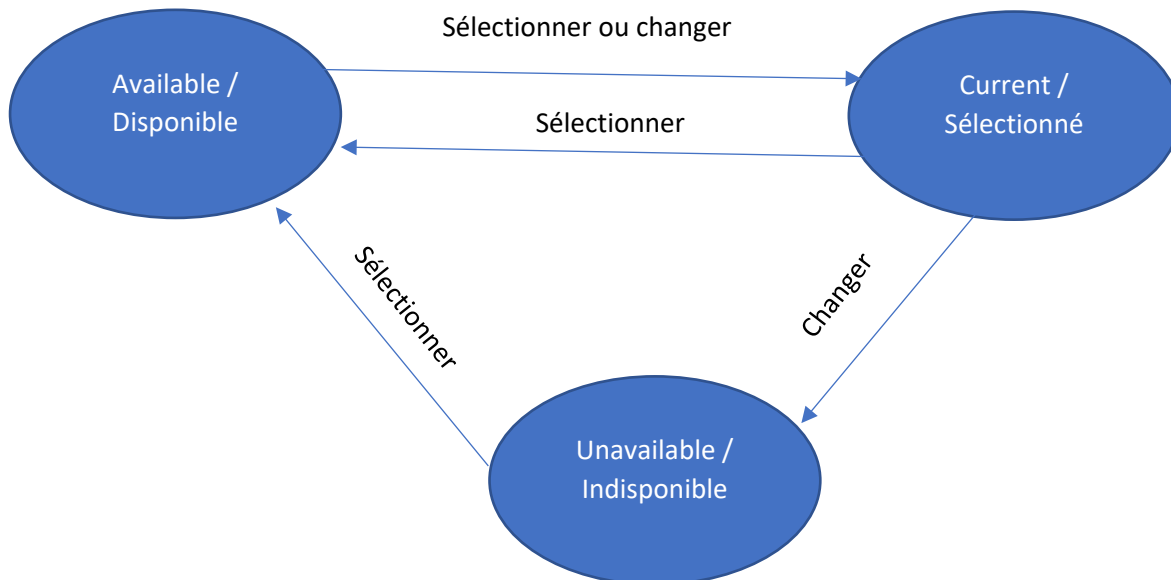
Afin de gérer l'ordre de passage d'un candidat, j'ai utilisé une variable priorité, qui sert de « poids ». Le candidat à une tâche avec le poids le plus élevé est choisi pour traiter la tâche. Une fois la tâche accomplie, son poids revient à zéro, l'amenant au début de la file d'attente.

A l'ajout d'un nouveau candidat, la priorité des candidats existants est incrémentée de 1 et le nouveau candidat a une priorité à zéro, l'amenant de fait en début de file.

Bien que l'implémentation le prévienne, il est possible que la priorité de deux candidats soit assignée à la même valeur manuellement. Si cela devait arriver, il a été convenu avec les clients qu'un des candidats serait choisi arbitrairement, le second étant désigné au prochain tour.

### *Automate à état fini*

Afin de cerner le besoin j'ai mis en place un automate à état fini afin de connaître les possibles états d'un candidat :



### *Implémentation*

Ci-dessous se trouve la méthode utilitaire traitant la rotation des candidats. La liste de candidats est passée en paramètre, pour assurer la réutilisabilité de la méthode. Cette méthode correspond à l'action « Sélectionner » de l'automate.

Une première itération est effectuée sur toute la liste. Pour chaque élément de la liste :

- La priorité est incrémentée. Un candidat avec une priorité de 0 passe à 1, la priorité 1 passe à 2, etc.
- Si le candidat est le candidat courant, son statut passe à « disponible ». Sa priorité est réassignée à 0.
- Si le candidat est indisponible, son statut passe à « disponible ».

Une fois les statuts mis à jour pour tous les candidats, le nouveau candidat est sélectionné. Pour cela, la liste est filtrée afin de n'obtenir que les candidats disponibles, puis, à l'aide d'un comparateur, le candidat ayant la plus haute priorité est sélectionné. Son statut est passé à « sélectionné ».

```

/**
 * Method that rotates {@code Candidate} using Round robin logic.
 *
 * @param candidates
 */
public static void rotate(Set<Candidate> candidates) {
    candidates.forEach(candidate -> {
        // increment priority of each candidate
        candidate.incrementPriority();
        if (candidate.getStatus().equals(Status.CURRENT)) {
            /*
             * given candidate status is set to "AVAILABLE" and its priority
             * is set to 0 to put him at the tail of the queue.
             */
            candidate.becomesPrevious();
        } else if (candidate.getStatus().equals(Status.UNAVAILABLE)) {
            // Unavailable candidates are reset to available each week
            candidate.setStatus(Status.AVAILABLE);
        }
    });
    /*
     * search for the first available candidate with the highest priority
     * and set his status to current
     */
    candidates.stream().filter(Candidate::isAvailable)
        .max(new PriorityComparator())
        .ifPresent(Candidate::becomesCurrent);
}

```

## Couche Serveur

### Généralités

Les constructeurs des entités et DTO ont tous, sauf besoin spécifique, une visibilité protégée, pour éviter leur instanciation et laisser le soin à Spring (via Jackson et JPA) d'instancier les objets.

```

/**
 * Empty no-arg empty constructor.
 */
protected CandidateCreateDto() {
    //
}

/**
 * Empty no-arg protected constructor
 */
protected MemberUpdateDto() {
    //
}

```

Les entités héritent d'une entité abstraite qui fournit un identifiant et une séquence d'incrémentation et les rend tous sérialisables :

```

@SuppressWarnings("serial")
@MappedSuperclass
public abstract class AbstractEntity implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
}

```

## Domaine

### Entités

Les entités métiers permettent le mapping objet-relationnel. L'avantage de l'ORM est qu'il fait le lien avec la base de données SQL qu'on utilise une base H2, MySQL, PostgreSQL... JPA permet de faire le lien par annotation afin de garder la maîtrise des nommages, notamment pour les index.

```
@Entity
@Table(name = "candidates", indexes = {
    @Index(name = "candidates_member_id_IDX", columnList = "member_id"),
    @Index(name = "candidates_task_id_IDX", columnList = "task_id")
}, uniqueConstraints = @UniqueConstraint(name = "candidates_member_id_task_id_UNIQUE", columnNames = {
    "member_id", "task_id" }))
public class Candidate extends AbstractEntity {
```

Cela permet également de gérer les contraintes en base de données (non nullité, taille) et de donner des types spécifiques (notamment des ENUM).

```
@ManyToOne
@JoinColumn(nullable = false, foreignKey = @ForeignKey(name = "candidates_member_id_FK"))
private Member member;

@Column(nullable = false, columnDefinition = "enum('AVAILABLE', 'UNAVAILABLE', 'CURRENT')")
@Enumerated(value = EnumType.STRING)
private Status status;

@JoinColumn(nullable = false, foreignKey = @ForeignKey(name = "candidates_task_id_FK"))
@ManyToOne
private Task task;

@Column(nullable = false, length = 4)
private int priority;
```

### DTO

```
public class CandidateCreateDto {

    @NotNull
    private EntityIdDto member;

    @NotNull
    private EntityIdDto task;

    private int priority = 0;

    private Status status = Status.AVAILABLE;
```

Le package domain contient également les DTO. Ces derniers permettent en outre la validation des données.

Si des objets servent de champ aux DTO, alors ces objets sont également des représentations sous forme de DTO.

Les DTO respectent une convention de nommage qui permet de connaître leur utilité : ils sont composés de l'entité visée suivie de l'action appliquée et suffixés par Dto.

## API

### Contrôleurs

Les contrôleurs font office de « porte d'entrée à l'API. Comme leur nom l'indique, ils contrôlent les entrées et sorties de l'application. Ce contrôle s'exerce aussi bien sur les droits d'accès (autorisation / authentification) que sur les données envoyées (programmation défensive).

Le contrôleur répond aux normes REST. Spring permet de l'identifier comme un contrôleur grâce à l'annotation `@RestController` ce qui permet que les appels passés au `DispatcherServlet` de Spring puissent être redirigés vers ce contrôleur (design pattern Front Controller) et que ces appels soient scannés par le `ControllerAdvice`. La redirection est faite grâce au mapping offert par l'annotation `@RequestMapping`.

```
@RestController
@RequestMapping("/candidates")
public class CandidateController {

    private final CandidateService candidateService;

    /**
     * Protected constructor to autowire needed bean.
     * <p>
     * injects {@code CandidateService} interface
     *
     * @param candidateService the injected {@code CandidateService}.
     */
    protected CandidateController(CandidateService candidateService) {
        this.candidateService = candidateService;
    }
}
```

J'ai choisi de rendre le constructeur protégé puisque l'instanciation à la main de ce contrôleur ne ferait pas sens.

De plus, Le service est injecté permettant d'appliquer la Dependency Inversion (un des principes SOLID).

L'exemple de endpoint <sup>18</sup> donné ci-dessous montre un POST<sup>19</sup>. L'entité à persister est reçue sous forme de DTO. Les données de cette entité sont validées grâce à l'annotation `@Valid`, qui viendra vérifier les contraintes indiquées dans le DTO.

Le point d'accès est sécurisé par l'annotation `@HasRoleUser` (voir implémentation -> Couche serveur -> sécurité).

```
/**
 * Persists a {@code CandidateCreateDto}.
 *
 * @param inputs the {@code candidateCreateDto} to persist.
 * @return the attributed id encapsulated in a {@code EntityIdDto}.
 */
@PostMapping
@HasRoleUser
public EntityIdDto save(@Valid @RequestBody CandidateCreateDto inputs) {
    return candidateService.save(inputs);
}
```

La seule action donnée à cette méthode est d'utiliser la méthode `save` du service.

Tout le reste (validation, sécurité) est fait par annotations (cf. supra).

## Services

Tous les services du projet étendent la classe abstraite `AbstractService`, qui fournit des méthodes utilitaires permettant notamment la transformation d'entités en DTO et inversement.

```
public <S, D> D convert(S source, Class<D> destination) {
    return modelMapper.map(source, destination);
}
```

Méthode permettant la transformation d'une entité en DTO ou inversement

```
public <S, D> void merge(S source, D destination) {
    modelMapper.map(source, destination);
}
```

Méthode permettant de fusionner deux objets : les données de la source sont ajoutées à l'objet de destination

<sup>18</sup> Point d'accès sous forme d'URL

<sup>19</sup> POST est une des méthodes http indiquant la persistance



```

public <S, D> List<D> convertSetToList(Set<S> source,
    Class<D> destination) {
    return source.stream()
        .map(elt -> convert(elt, destination))
        .collect(Collectors.toList());
}

```

Méthode permettant de convertir tous les objets d'une liste de type Set et de retourner un nouveau set.

Les services sont implémentés en deux temps : une interface déclarant les méthodes abstraites et une implémentation de cette interface. Cela permet en cas de besoin de créer une seconde implémentation pour des besoins spécifiques. Bien que le besoin ne se soit pas présenté dans Sprint Planner, j'ai choisi de respecter ce design qui est une bonne pratique. Le fait d'avoir une interface constitue un contrat pour les développeurs qui travailleraient sur le projet.

```

@Service
public class CandidateServiceImpl extends AbstractService
    implements CandidateService {

    private final CandidateJpaRepository candidateRepository;
}

```

Le repository est injecté par Spring. Le fait de le déclarer final permet d'interdire un ré-assignement de d'assurer ainsi le fonctionnement attendu.

```

@Override
public EntityIdDto save(CandidateCreateDto inputs) {
    Long taskId = inputs.getTask().getId();
    List<Candidate> candidates = candidateRepository
        .findAllByTaskId(taskId);
    candidates.forEach(Candidate::incrementPriority);
    Candidate candidate = convert(inputs, Candidate.class);
    candidates.add(candidate);
    candidateRepository.saveAll(candidates);
    return convert(candidate, EntityIdDto.class);
}

```

La création d'un candidat entraîne l'incrémement des priorités. Il faut donc :

- Rechercher tous les candidats à la tâche donnée
- Incrémenter leur priorité
- Ajouter le nouveau candidat à la liste des candidats
- Sauvegarder la liste avec le nouveau candidat
- 

## Repositories

JPA fournit une gestion automatique des connexions JDBC<sup>20</sup> et des méthodes prédéfinies (save, findOne, findAll, ...) via des interfaces. Cela facilite grandement le travail en réduisant le code à écrire. Pour en bénéficier, tous les repositories de Sprint Planner étendent l'interface JpaRepository.

```

public interface CandidateJpaRepository extends JpaRepository<Candidate, Long>

```

Pour les requêtes plus spécifiques, j'ai majoritairement utilisé Spring Data JPA qui fournit les requêtes dérivées. Cela permet par convention de nommage d'écrire un équivalent de requête SQL que Spring va traduite en JPQL<sup>21</sup>.

```

List<Candidate> findAllByTaskName(String taskName);

```

Pour quelques requêtes très spécifiques (avec un attribut optionnel par exemple), j'ai préféré une requête JPQL rédigée manuellement.

<sup>20</sup> Java Data Base Connection

<sup>21</sup> Java Persistence Query Language

```

/**
 * Retrieve a {@code Candidate} by its task, shift and status.
 * <p>
 * shift is optional. If not provided, the {@code Candidate} will be
 * retrieved by its task and status.
 * <p>
 */
// This method could be written with {@code Spring JPA} derived queries, but
// a standard query is more readable
static final String FIRST_ELIGIBLE_CANDIDATE = "select c from Member m join Candidate c on m.id = c.member "
    + "join Task t on t.id = c.task where t.name = :task and c.status = :status and (:shift is null "
    + "or m.shift = :shift) order by c.priority desc";

@Query(JpqlQuery.FIRST_ELIGIBLE_CANDIDATE)
List<Candidate> findFirstCandidateByParameters(
    @Param("task") String task,
    @Param("status") Status status,
    @Param("shift") Shift shift,
    Pageable pageable);

```

La requête est externalisée dans un fichier de constantes JpqlQuery, afin d'assurer une meilleure lisibilité et de permettre la documentation de ladite requête.

## Sécurité

### Authentification et autorisations

L'authentification est effectuée par jeton JWT. L'implémentation compte plusieurs classes de configuration permettant de gérer le jeton :

```

public class CustomTokenEnhancer implements TokenEnhancer {
    static final String USER_ID_KEY = "userId";

    @Override
    public OAuth2AccessToken enhance(OAuth2AccessToken accessToken,
        OAuth2Authentication authentication) {
        Map<String, Object> additionalInfo = new HashMap<>();
        CustomUserDetails user =
            (CustomUserDetails) authentication.getPrincipal();
        additionalInfo.put(USER_ID_KEY, user.getId());
        ((DefaultOAuth2AccessToken) accessToken)
            .setAdditionalInformation(additionalInfo);
        return accessToken;
    }
}

@Bean
protected JwtAccessTokenConverter accessTokenConverter() {
    JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
    Resource resource = new ClassPathResource(keyStore);
    char[] password = keyPass.toCharArray();
    KeyStoreKeyFactory factory = new KeyStoreKeyFactory(resource, password);
    converter.setKeyPair(factory.getKeyPair(keyAlias));
    converter.setAccessTokenConverter(customAccessTokenConverter);
    return converter;
}

```

Le tokenEnhancer permet de configurer le jeton avec des informations additionnelles.

Ici, le UserDetails fourni par Spring étant modifié, l'enhancer ajoute au jeton l'identifiant de l'utilisateur.

Le convertisseur de jeton d'accès permet de configurer les identifiants qui seront utilisés pour vérifier la validité du jeton.

La configuration globale du serveur d'autorisation est faite par la méthode configure ci-dessous : les configurations du jeton (ci-dessus) sont passés au TokenChainEnhancer. Ce dernier est passé au configureur avec l'authenticationManager (qui gèrera la requête) et le userDetailsService (service par lequel passera la requête).

```

@Override
public void configure(AuthorizationServerEndpointsConfigurer configurator)
    throws Exception {
    TokenEnhancerChain tokenEnhancerChain = new TokenEnhancerChain();
    tokenEnhancerChain.setTokenEnhancers(
        Arrays.asList(tokenEnhancer(), accessTokenConverter()));
    configurator.tokenStore(tokenStore()).tokenEnhancer(tokenEnhancerChain)
        .authenticationManager(authenticationManager)
        .userDetailsService(userDetailsService);
}

```

La gestion des autorisations sur les différents endpoints est faite par annotation. Afin d'améliorer la lisibilité, j'ai créé des annotations personnalisées :

```
@PostMapping
@HasRoleUser
public EntityIdDto save(@Valid @Req
    return candidateService.save(in
}
```

L'annotation @HasRoleUser signifie que chaque personne qui a à minima un rôle USER peut atteindre ce endpoint. Dans les faits, cela signifie qu'un USER ou un ADMIN pourra atteindre le endpoint.

L'annotation @HasRoleUser encapsule l'annotation @PreAuthorize de Spring:

```
@PreAuthorize("hasRole('ROLE_USER') or hasRole('ROLE_ADMIN')")
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target({ ElementType.METHOD, ElementType.TYPE })
public @interface HasRoleUser {
    // no action, encapsulate PreAuthorized roles for readability
}
```

Protection contre les failles

- Injection SQL

Pour prévenir l'injection SQL, j'ai proscrit les chaînes concaténées pour les requêtes en base de données. Cela permet de se protéger à moindre coût contre l'injection SQL.

Spring data JPA permet d'écrire des méthodes qui reflètent une requête SQL en insérant les variables de façon à prévenir l'injection SQL.

```
List<Candidate> findAllByTaskName(String taskName);
```

La requête ci-dessus est interprétée comme une requête JPQL et prévient l'injection SQL.

Lorsque l'utilisation de requêtes écrites est nécessaire, j'ai injecté les variables à l'aide de la syntaxe « :variable ». Ainsi, JPA insère la variable dans la requête en prévenant l'injection :

```
from Member m join Candidate c on m.id = c.member join Task t on t.id = c.task
where t.name = :taskName and c.status = :status";
```

- Cross Site request Forgery

S'agissant d'une application interne, utilisable uniquement sous le VPN de l'entreprise et ne traitant pas de données sensibles, la protection contre les failles CSRF n'est pas une priorité absolue.

De plus, la création d'un token CSRF pour vérifier la ressource appelante ferait de sprint Planner une application stateful alors que le but est qu'elle reste stateless, afin de laisser aux utilisateurs le choix du déploiement (sur un ou plusieurs serveurs).

Une autre solution serait de requêter le token CSRF à chaque requête de type POST / PATCH / PUT / DELETE. Cela permettrait de conserver une application stateless, au détriment du nombre d'appels entre le front end et le back end. La mise en place d'une telle solution est en contradiction avec l'utilisation d'un token JWT pour l'authentification, qui sert précisément à limiter les aller-retours en conservant les données d'authentification avec les rôles.

Comme nous le verrons dans la partie sur la veille, il existe une solution alternative, qui ne répond pas entièrement au besoin mais a l'avantage de sécuriser en partie l'application contre les attaques de type CSRF. Il s'agit de la restriction des appels à une origine.

Cette solution m'a paru répondre au besoin tel qu'indiqué ci-dessus tout en maintenant une architecture stateless sans multiplier les requêtes :

```
@Value("${config.sprintplanner.allowedOrigin}")
private String allowedOrigin;

@Override
public void doFilter(ServletRequest servletRequest,
    ServletResponse servletResponse, FilterChain chain)
    throws IOException, ServletException {
    HttpServletResponse response = (HttpServletResponse) servletResponse;
    response.setHeader("Access-Control-Allow-Origin", allowedOrigin);
    response.setHeader("Access-Control-Allow-Credentials", "true");
    response.setHeader("Access-Control-Allow-Headers",
        "x-requested-with, Authorization, Content-Type");
    response.setHeader("Access-Control-Allow-Methods", "*");
    chain.doFilter(servletRequest, servletResponse);
}
```

La mise en place d'une origine autorisée permet de limiter les appels à cette seule origine. Ici, l'origine est récupérée dans l'application.yml. Au déploiement, cette variable peut simplement être injectée par Jenkins, XL-Deploy ou même un script de démarrage en Shell.

## Couche de présentation

La couche de présentation étant implémenté avec Angular, elle respecte de fait l'approche MVC<sup>22</sup>.

### Modèle

La mise en application du MVC implique que la couche de présentation ait un modèle des données qui seront manipulées.

```
import { IdDto } from './IdDto.model';

export class CandidateCreateDto {
    member: IdDto;
    task: IdDto;

    constructor(member: IdDto, task: IdDto) {
        this.member = member;
        this.task = task;
    }
}
```

Le modèle ci-contre représente la création d'un candidat.

S'agissant d'un DTO, son nommage suit les conventions indiquées précédemment.

Il est le reflet du DTO de la couche serveur.

---

<sup>22</sup> Model – View – Controller

## Vue

Angular permet la mise en place de templates et leur réutilisation. Ainsi, dans le cas de la gestion des candidats, un composant partagé est créé, puis réutilisé pour chaque tâche (releaser, support, tester). La création d'une nouvelle tâche est alors simple et rapide.

```
<div [collapse]="isCandidateCollapsed">
  <div class="row text-center info-part">
    <div class="col-sm-12">
      <ag-grid-angular
        #agGrid
        class="ag-theme-material"
        [rowData]="rowData"
        [gridOptions]="gridOptions"
        autoSize=true
        style="width:100%;"
        domLayout="autoHeight"
        [frameworkComponents]="frameworkComponents"
        [overlayNoRowsTemplate]="overlayNoRowsTemplate"
        (cellValueChanged)="onCellValueChanged($event)">
      </ag-grid-angular>
```

Grille d'affichage des candidats en HTML.

Alors que l'utilisation d'AgGrid permet que la grille soit gérée dynamiquement, bootstrap fournit des classes permettant un affichage responsif (col-sm-12).

Les données sont passées via la variable rowData qui sera déterminée dans la partie controller du modèle MVC.

Ainsi, l'affichage dépendra des données envoyées par le controller et donc de la route prise par l'utilisateur.

L'affichage effectif de la grille des candidats et du candidat courant est effectué avec un simple template HTML qui reprend la vue ci-dessus. La tâche est indiquée dans la balise appelant le composant (task). Selon la tâche renseignée, le controller récupérera les données adéquates.

```
<app-page-header [title]="title" [subtitle]="subtitle"></app-page-header>

<div class="container section">
  <app-manage-candidate task="releaser"></app-manage-candidate>
</div>
```

La gestion des candidats par shift (Paris ou Bangalore) bénéficie également de ce design. L'ajout du shift permet de déterminer si les données concernent Paris ou Bangalore. Le shift est quant à lui déterminé par le controller grâce à la route utilisée (e.g. : .../supports/bangalore).

```
<app-page-header [title]="title" [subtitle]="subtitle"></app-page-header>

<div class="container section">
  <app-manage-candidate task="technical" [shift]="shift"></app-manage-candidate>
  <app-manage-candidate task="functional" [shift]="shift"></app-manage-candidate>
</div>
```

L'application devant répondre à des obligations de charte graphique, j'ai passé du temps à naviguer sur l'application RiFT de mon équipe, pour me familiariser avec son style et le reproduire.

```
.btn {
  color: white;
  background-color: darkred;
  border-color: darkred;
  margin: 1% 1%;
  max-width: 40%;
```

Concernant les styles, une feuille de style est mise en place de façon globale pour toute l'application.

Cela permet d'avoir un style commun, réutilisé dans toute l'application.

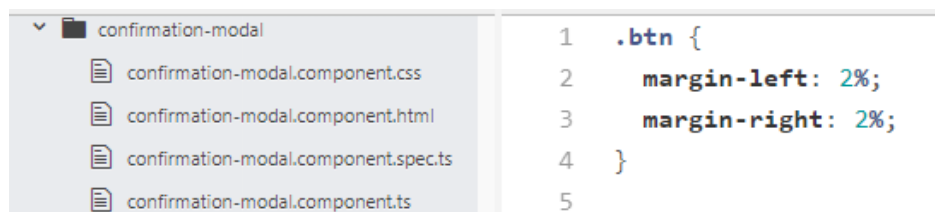
1

D'autres styles fournis par les librairies sont utilisés et donc importés dans le style global :

```
@import "~ag-grid-community/dist/styles/ag-grid.css";
@import "~ag-grid-community/dist/styles/ag-theme-balham.css";
@import '~ag-grid-community/dist/styles/ag-theme-material.css';
@import "app/shared/styles/ag-grid.css";
```

La dernière ligne correspond à un style que j'ai écrit pour surcharger les styles d'AgGrid, afin de répondre aux exigences de la charte graphique imposée.

Enfin, chaque composant est doté d'un fichier de style personnel. Si la majorité n'a pas besoin de style spécifique, certains composants comme les modales nécessitent un style propre :



## Contrôleurs

Les contrôleurs font le lien avec le serveur. Ils renseignent les données qui seront rendues par la vue.

```
ngOnInit() {
  this.taskTitle = this.getTaskTitle();
  this.getCandidates();
  this.getNonCandidates();
  this.getTask();
}
```

A l'initialisation, le contrôleur requête le serveur afin de récupérer les candidats et les non-candidats à la tâche.

La première requête permettra d'afficher la liste des candidats et le candidat courant, tandis que la seconde servira pour le menu déroulant d'ajout de candidats.

La méthode `getCandidates()` ci-dessous permet de récupérer la liste de tous les candidats à une tâche. Cette liste est ensuite traitée afin de récupérer les candidats à afficher dans le tableau (`rowData`, cf. supra) et le candidat actuellement en charge de la tâche.

```
public getCandidates() {
  this.candidateService.getCandidates(this.task, this.shift)
    .subscribe((candidates: Candidate[]) => {
      this.rowData = candidates;
      this.currentCandidate = candidates.find(candidate => candidate.status === Status.CURRENT);
      this.currentCandidateName =
        this.currentCandidate.member.firstname + " " + this.currentCandidate.member.lastname;
    });
}
```

## Sécurité

### Authentification et autorisations

Les routes de la couche client sont sécurisées à l'aide d'un service déterminant si l'utilisateur a les droits nécessaires à l'accès. Le jeton JWT (voir partie client) est décrypté et contient les autorisations de l'utilisateur. L'intérêt de mettre en place un jeton sous forme de cookie est de spécifier une

expiration. Ainsi, l'utilisateur peut quitter et revenir sur l'application sans forcément avoir à s'authentifier de nouveau comme ce serait le cas avec un session storage, mais il est possible de donner une validité de 24h au jeton afin de gérer les révocations de droits.

```
export const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'members', component: MemberComponent, canActivate: [AuthGuard], data: { roles: ["ROLE_ADMIN", "ROLE_USER"] } },
  { path: 'releasers', component: ReleaserComponent, canActivate: [AuthGuard], data: { roles: ["ROLE_ADMIN", "ROLE_USER"] } },
  { path: 'testers', component: TesterComponent, canActivate: [AuthGuard], data: { roles: ["ROLE_ADMIN", "ROLE_USER"] } },
  { path: 'versions', component: ReleaseVersionComponent, canActivate: [AuthGuard], data: { roles: ["ROLE_ADMIN", "ROLE_USER"] } },
  { path: 'issues', component: IssueReconciliationComponent, canActivate: [AuthGuard], data: { roles: ["ROLE_ADMIN", "ROLE_USER"] } },
  { path: 'supports/bangalore', component: SupportComponent, canActivate: [AuthGuard], data: { roles: ["ROLE_ADMIN", "ROLE_USER"] } },
  { path: 'supports/paris', component: SupportComponent, canActivate: [AuthGuard], data: { roles: ["ROLE_ADMIN", "ROLE_USER"] } },
  { path: 'configuration', component: ConfigurationComponent, canActivate: [AuthGuard], data: { roles: ["ROLE_ADMIN"] } }
];
```

Chaque route est sécurisée en cohérence avec les autorisations du serveur. Si un utilisateur tente d'accéder même manuellement à une ressource à laquelle il n'a pas accès, il est redirigé vers la page d'accueil.

### Protection contre les failles

Angular est construit avec une protection contre les failles XSS en considérant qu'aucun input n'est de confiance (untrusted). De plus, la compilation de la couche client est faite avant déploiement (hors ligne), ce qui permet d'éviter l'injection de templates et renforce ainsi la sécurité contre les attaques XSS.

#### Angular's cross-site scripting security model

To systematically block XSS bugs, Angular treats all values as untrusted by default. When a value is inserted into the DOM from a template, via property, attribute, style, class binding, or interpolation, Angular sanitizes and escapes untrusted values.

*Angular templates are the same as executable code:* HTML, attributes, and binding expressions (but not the values bound) in templates are trusted to be safe. This means that applications must prevent values that an attacker can control from ever making it into the source code of a template. Never generate template source code by concatenating user input and templates. To prevent these vulnerabilities, use the [offline template compiler](#), also known as *template injection*.

Source : documentation angular

## TESTS

### Conventions

Les classes de test portent le nom de la classe qu'elles testent, suffixé par « Test ». Dans la couche serveur, elles sont rangées dans le package `src/test/java` et dans des sous packages qui dupliquent les packages du projet. Ainsi, une classe de test est trouvable dans le même package que la classe testée, à partir de `src/test/java` au lieu de `src/main/java`.

Les composants de la couche client étant rangés dans des dossiers propres (convention Angular), le dossier d'un composant contient la classe de test.

Les jeux de test ont une convention de nommage propre : les noms des tests sont en `snake_case`. Cela permet de nommer le test par une phrase complète, ce qui rend le test parfaitement compréhensible.

Par convention, le nommage de la méthode de test commence par « `should_` » puis explicite clairement l'attendu.

### Configuration

Les tests unitaires et d'intégration bénéficient de classes utilitaires. Ces classes sont abstraites puisqu'il n'y a pas lieu de les instancier. Leurs méthodes sont donc statiques, pour être invoquées depuis les classes en héritant.

#### *Tests unitaires*

J'ai écrit une classe de configuration qui est étendue par tous les tests unitaires, afin qu'ils bénéficient des objets et méthodes nécessaires.

Les tests unitaires utilisent un `ObjectMapper`, objet fourni par Jackson pour sérialiser et désérialiser.

L'`ObjectMapper` est normalement instancié en tant que Bean dans le contexte Spring lors du lancement de l'application. L'application n'étant pas lancée dans le cadre des tests unitaires, il convient de déclarer un `ObjectMapper` dûment configuré pour l'exécution desdits tests.

Une méthode de setup est donc créée à l'aide de l'annotation `@BeforeAll`<sup>23</sup>.

```
@BeforeAll
protected static void setUp() {
    MAPPER.setVisibility(
        MAPPER.getSerializationConfig().getDefaultVisibilityChecker()
            .withFieldVisibility(JsonAutoDetect.Visibility.ANY)
            .withGetterVisibility(JsonAutoDetect.Visibility.NONE)
            .withIsGetterVisibility(JsonAutoDetect.Visibility.NONE)
            .withSetterVisibility(JsonAutoDetect.Visibility.NONE));
    MAPPER.registerModule(new JavaTimeModule());
    MAPPER.configure(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS, false);
}
```

La méthode générique ci-dessous permet de transformer un json en objet par l'utilisation de l'`ObjectMapper`. Cela permet de rendre beaucoup plus facile et lisible la conversion.

---

<sup>23</sup> Annotation permettant que la méthode annotée soit exécutée une fois avant l'exécution de tous les tests d'une classe de test.



```
protected final <D> D jsonConvert(String inputs, Class<D> destinationType) {
    D converted = null;
    try {
        converted = MAPPER.readValue(inputs, destinationType);
    } catch (IOException ex) {
        throw new IllegalArgumentException(
            "wrong json format in csv source file", ex);
    }
    return converted;
}
```

### Tests d'intégration

Les tests d'intégration sont effectués en lançant l'application. Cette dernière est lancée avec un contexte de test, afin de bénéficier d'une configuration propre (base in-memory<sup>24</sup>, port serveur dédié, logs configurés pour les tests, déclaration du script d'insertion de données ...).

```
@ExtendWith(SpringExtension.class)
@SpringBootTest(webEnvironment = WebEnvironment.DEFINED_PORT)
@ActiveProfiles(profiles = "test")
public abstract class SetupIntTest {
```

Les beans utiles sont injectés (ModelMapper, ObjectMapper et LocalValidatorFactoryBean).

Outre les méthodes utilitaires permettant la conversion d'objets, une méthode permet de vérifier la validité des données en entrée :

```
protected final <D> boolean isValid(D inputs) {
    Validator validator = validatorFactory.getValidator();
    Set<ConstraintViolation<D>> violations = validator.validate(inputs);
    return violations.isEmpty();
}
```

Les données qui serviront sont insérées dans la base in-memory par le biais d'un script DML dédié, exécuté au démarrage de l'application test.

### Tests utilisateurs - UAT<sup>25</sup>

Afin de s'assurer que l'application réponde aux besoins utilisateurs, j'ai mis en place des tests dits UAT. Pour leur réalisation, un fichier Excel est édité et l'application est donnée en test aux utilisateurs. Ainsi, ils peuvent à la fois manipuler l'application et faire un retour sur son ergonomie et surtout valider ou non les scénarios décrits.

## Exemples d'implémentation

### Test unitaire de l'algorithme Round Robin

Une liste de candidats est créée avant chaque test à l'aide de l'annotation @BeforeEach<sup>26</sup>, afin de garder des données cohérentes et identiques pour chaque jeu de test.

Les données sont les suivantes :

<sup>24</sup> Base de données non persistée, qui n'existe que pendant que l'application est lancée

<sup>25</sup> User Acceptance Tests

<sup>26</sup> Annotation Junit permettant que la méthode annotée soit exécutée avant chaque test

```

@BeforeEach
void setListUp() {
    unavailable = jsonConvert(
        "{\"member\":{\"id\":-2}, \"task\":{\"id\":-4},\"status\":\"UNAVAILABLE\", \"priority\":1}",
        Candidate.class);
    available = jsonConvert(
        "{\"member\":{\"id\":-3}, \"task\":{\"id\":-4},\"status\":\"AVAILABLE\", \"priority\":2}",
        Candidate.class);
    current = jsonConvert(
        "{\"member\":{\"id\":-4}, \"task\":{\"id\":-4},\"status\":\"CURRENT\", \"priority\":3}",
        Candidate.class);
    candidates = Sets.newHashSet(unavailable,
        available,
        current);
}

```

Le premier test assure la modification des priorités lors de la rotation des candidats :

```

@Test
void should_change_priorities() {
    RoundRobinHandler.rotate(candidates);
    Assertions.assertThat(
        () -> Assertions.assertEquals(unavailable.getPriority(), 2),
        () -> Assertions.assertEquals(current.getPriority(), 0),
        () -> Assertions.assertEquals(available.getPriority(), 3));
}

```

Le second test assure que les statuts sont bien mis à jour :

```

@Test
void should_change_statuses() {
    RoundRobinHandler.rotate(candidates);
    Assertions.assertThat(
        () -> Assertions.assertEquals(unavailable.getStatus(),
            Status.AVAILABLE),
        () -> Assertions.assertEquals(current.getStatus(),
            Status.AVAILABLE),
        () -> Assertions.assertEquals(available.getStatus(),
            Status.CURRENT));
}

```

### Test d'intégration de la gestion d'un candidat

Les tests d'intégration permettent de tester le bon fonctionnement du lien entre la partie serveur et la partie base de données.

Ci-dessous, un test de création de candidat. Il est à noter que l'utilisateur habilité est simulé par l'annotation `@WithMockUser`<sup>27</sup>.

```

@ParameterizedTest
@CsvFileSource(resources = "/candidateCreation.csv", delimiter = ';')
@WithMockUser(username = "admin", password = "pwd", roles = "ADMIN")
void should_save_new_candidate(String json) {
    CandidateCreateDto dto = jsonConvert(json, CandidateCreateDto.class);
    EntityIdDto actual = controller.save(dto);
    assertNotNull(actual);
}

```

<sup>27</sup> Annotation Spring Security Test qui permet de simuler un utilisateur avec un rôle spécifique

Les tests d'intégration permettent également de tester le bon fonctionnement des autorités, en modifiant simplement le rôle de l'utilisateur simulé. Dans l'exemple ci-dessous, l'utilisateur n'ayant pas les droits pour traiter l'opération, il doit rencontrer une exception :

```
@ParameterizedTest
@CsvFileSource(resources = "/candidateDelete.csv", delimiter = ';')
@WithMockUser(username = "usr", password = "pwd", roles = "USER")
void should_throw_access_denied(String json) {
    CandidateDeleteDto dto = jsonConvert(json, CandidateDeleteDto.class);
    Assertions.assertThrows(AccessDeniedException.class,
        () -> controller.delete(dto));
}
```

Enfin, les tests d'intégration permettent de vérifier la bonne validation des données d'entrée :

```
@ParameterizedTest
@CsvFileSource(resources = "/invalidCandidateCreation.csv", delimiter = ';')
void should_fail_validation_when_creating(String json) {
    CandidateCreateDto tested = jsonConvert(json, CandidateCreateDto.class);
    assertFalse(isValid(tested));
}
```

## Tests UAT

Ci-dessous, un exemple de test UAT pour le cas d'un ajout de nouveau candidat.

User Acceptance Tests			
Scenario	expected	Success / failure	Observations
As an authenticated user, I want to add a single member to the releaser candidate list: - I navigate to releasers page - I select a member in the dropdown list	add given member to releaser candidate list	SUCCESS	
	created member should be available with priority of 0	SUCCESS	
	increment other members priority	SUCCESS	

Un scénario est décrit, puis sont décrits les attendus et le statut (succès / échec). L'utilisateur peut faire une observation (en cas d'échec ou en cas de remarque sur l'ergonomie). Cela me sert alors à corriger un bug éventuel et à améliorer l'expérience utilisateur.

## Tests de non-régression

Les tests unitaires, d'intégration et UAT sont rejoués avant chaque nouveau déploiement. Cette batterie de tests sert de test de non-régression : si un test devait échouer, il y a régression et le déploiement ne pourra être fait qu'une fois la régression corrigée et la validation des tests.

# VEILLE ET RECHERCHES

## VEILLE SUR LE SUJET DE LA SECURITE

### Méthode de veille

J'effectue ma veille technologique de plusieurs façons : la première grâce à l'algorithme de Google. Ce dernier me propose des informations en relation avec l'informatique en analysant mes recherches qui portent majoritairement sur ce sujet.

Ensuite, je me forme à l'aide de sites sur différents sujets. Mon entreprise m'a par exemple doté d'un abonnement sur Pluralsight, un site de formation par vidéos. Dans la mesure du possible, je consacre une heure de recherche/apprentissage par jour (dans le train essentiellement).

Enfin, j'effectue des recherches ponctuelles sur des sujets divers, notamment la sécurité. Pour ce sujet spécifique, ma source de prédilection est le site de la fondation OWASP<sup>28</sup>.

### XXE – XML External Entity

C'est la veille sur le top 10 des failles OWASP qui m'a permis de découvrir cette faille. Cette faille a été ajoutée au top 10 d'OWASP en 2017 et consiste à récupérer des ressources par l'utilisation du XML.

#### *Définition*

Le principe consiste à exploiter les entités externes du XML pour « variabiliser » des fichiers du serveur. L'envoi d'une requête de ce type retourne une réponse qui affiche les données du fichier passé en variable (par exemple `classpath:/application.yml`, qui renvoie le contenu du fichier `application.yml`).

#### *Analyse de mon code*

La découverte de cette faille m'a permis de chercher la vulnérabilité au niveau de mon serveur et de trouver une parade. En l'occurrence, n'exploitant pas le XML et traitant de données sous forme de DTO, je ne suis pas à proprement parler vulnérable aux failles XXE. Cependant, la simplicité d'implémentation pour régler la faille et l'incertitude liée aux développements futurs m'ont poussé à prendre les dispositions nécessaires.

#### *Correction*

La correction de cette faille se fait simplement sur un backend JAVA comme le mien : seul l'ObjectMapper rend le backend vulnérable. Or, la librairie Jackson fournit un builder qui permet la construction d'un ObjectMapper déjà configuré pour se prémunir des XXE : alors que l'instanciation de l'objectMapper avec un `new ObjectMapper()` ouvre la porte à cette faille, une instanciation comme ci-dessous est sans danger :

```
ObjectMapper mapper = Jackson2ObjectMapperBuilder.json().build();
```

---

<sup>28</sup> Open Web Application Security Project

## Veille sur les failles CSRF

La veille ne doit pas se cantonner à la recherche des nouveautés mais aussi à l'analyse de l'ancien. En effet, comme vu précédemment, une nouvelle faille a fait son apparition en 2017 dans le top 10 d'OWASP. Cependant, au même moment, une faille a été supprimée de ce top 10 car très peu de sites existants y sont maintenant vulnérables (5% selon OWASP).

Pourtant, cette faille peut s'avérer dangereuse selon les données et les objectifs de l'application visée : il s'agit du Cross Site Request Forgery.

### *Définition*

Le principe du CSRF<sup>29</sup> est de construire (forger) une requête et de la faire transmettre par un utilisateur à son insu au back-end attaqué. Pour cela, l'attaquant prépare une requête à envoyer au site visé et pousse la victime à soumettre cette requête. Cela peut se traduire par un lien, une demande d'authentification sur un site miroir, l'ouverture d'une pièce jointe, ...

La requête est envoyée au backend en bénéficiant du contexte d'authentification de la victime et le backend, ne pouvant différencier la requête d'une requête normale, exécute le traitement.

### *Sécurisation*

La sécurisation de cette faille se fait au moins de deux solutions :

La première consiste à utiliser le CORS fourni par certains navigateurs. En définissant quelle origine est autorisée à traiter avec le serveur, cela bloque de fait les requêtes venant de l'origine malveillante. Cela constitue une première barrière mais n'est pas une solution complète. En effet, les règles CORS n'ont pas été créées à des fins de sécurité, elle ne répond au besoin que pour les navigateurs qui l'appliquent puisque certains n'utilisent pas de requête dites « preflight » pour certains types de requête.

La seconde solution, à coupler à la première, est de mettre en place un jeton CSRF : seul le front end autorisé aura la possibilité d'obtenir ce jeton, qui servira de secret entre le backend et le front end pour valider les requêtes. Ainsi, un assaillant ne pourra pas envoyer une requête sans avoir le secret.

### *Analyse de mon code*

Comme indiqué dans la partie Implémentation > Couche Serveur > Sécurité > protection contre les failles, la mise en place d'un jeton CSRF n'est pas une priorité absolue. Elle représente un « nice-to-have » et pourra être implémentée dans le futur.

---

<sup>29</sup> Parfois XSRF pour Cross Site Request Forgery

## SITUATION DE TRAVAIL AYANT NECESSITE UNE RECHERCHE

### Fonctionnement de JAVAMAIL

Une des premières choses que j'ai eu à rechercher sur internet a été l'implémentation de l'envoi de mail. J'ai commencé la recherche en tapant simplement « send mail with java ».

Je suis rapidement tombé sur des solutions, mais aucune satisfaisante car elles ne respectaient pas le principe KISS selon moi : elles me semblaient trop complexes (e.g. : annexe 3).

J'ai donc cherché dans les ressources trouvées non pas la solution, mais les éléments importants : MIME Message, propriétés configurées et surtout la librairie utilisée (JAVAMAIL).

La seconde recherche a été un peu plus précise. Je savais que je cherchais des ressources sur JAVAMAIL, sur Spring Boot. J'ai donc recherché les deux éléments suivants :

*javamail javadoc java 8*

OVERVIEW	PACKAGE	CLASS	USE	TREE	DEPRECATED	INDEX	HELP
PREV PACKAGE	NEXT PACKAGE	FRAMES	NO FRAMES	ALL CLASSES			

Le résultat de la recherche m'a donné le lien vers la documentation JEE<sup>30</sup> 8

### Package javax.mail

The JavaMail™ API provides classes that model a mail system.

See: [Description](#)

*Javamail spring doc*

OVERVIEW	PACKAGE	CLASS	USE	TREE	DEPRECATED
PREV CLASS	NEXT CLASS	FRAMES NO FRAMES			
SUMMARY: NESTED		FIELD	CONSTR	METHOD	DETAIL: F

org.springframework.mail.javamail

### Interface JavaMailSender

All Superinterfaces:

MailSender

All Known Implementing Classes:

JavaMailSenderImpl

J'avais alors tous les éléments pour comprendre le fonctionnement de javamail et du javamail sender.

Avec les exemples trouvés précédemment et la documentation de JEE et Spring, j'ai pu créer deux classes pour l'envoi de mail : une classe de configuration pour renseigner les propriétés et un service qui effectue l'envoi (annexe 3 : Implémentation finale).

---

<sup>30</sup> Java Enterprise Edition

# CONCLUSION



## BILAN

### Appréciation

L'application est maintenant en production depuis plusieurs mois. De nombreuses modifications ont été apportées pour donner suite aux demandes client afin de répondre au mieux au besoin et d'adapter les fonctionnalités. Nous pouvons affirmer que l'application est mature et répond bien aux besoins. Elle est utilisée de façon courante et a bien été intégrée dans le process d'organisation de l'équipe.

### *Satisfactions*

Ma première satisfaction liée développement de cette application et à l'alternance en générale est le fait que j'ai été validé en tant que développeur par mon entreprise. Mon contrat a été signé en juillet 2020 au sein de l'équipe qui m'a accueilli en alternance.

L'application Sprint-planner est utilisée de façon courante par l'équipe ce qui est une grande satisfaction. Il est plaisant de savoir que le projet n'a pas juste été mis en place pour le titre RNCP mais pour répondre à un besoin, qu'il parvient à y répondre et que mes utilisateurs sont satisfaits.

Enfin, j'ai pu apprendre énormément dans le cadre de mon alternance et de mon projet, que ce soit au niveau front end (découverte d'Angular, du modèle MVC), back end (Java, Spring, Maven), base de données mais aussi au niveau devops (Jenkins, Sonar, Maven, XL-Deploy). J'ai d'ailleurs eu la chance de passer avec succès une certification Agile et la certification OPQUAST.

### *Difficultés rencontrées*

La principale difficulté a été de m'adapter à un besoin changeant. En effet, l'application étant une nouveauté, les clients me faisaient des retours réguliers qui modifiaient le besoin à mesure des livraisons, pour aller vers une application au plus près de leur organisation. Le fait de procéder en mode agile m'a néanmoins permis de modifier rapidement les livrables afin de répondre à ces modifications. Ainsi, ce qui a été une difficulté en début de développement c'est finalement avéré être très utile pour répondre au mieux aux souhaits des clients.

### Evolutions à venir

La prochaine fonctionnalité que je souhaite apporter à l'application consistera à suivre l'avancée d'un déploiement. L'objectif est de mettre en place par web socket un suivi des traitements jenkins, XL-Deploy, des exécutions de script, afin d'afficher en temps réel l'évolution du déploiement en cours.

Concernant l'utilisation de l'application, je souhaite déployer Sprint Planner à d'autres équipes afin d'avoir d'autres avis et ainsi améliorer l'application en continu.

Enfin, à titre personnel, je souhaite après le titre RNCP passer la certification Java.

## REMERCIEMENTS

Pour ses connaissances, ses qualités de formateur, son soutien, sa disponibilité et sa patience, un grand merci à Frank Marshall.

Merci à la société Générale et à Simplon pour leur confiance et leur accompagnement.

Pour m'avoir accueilli, m'avoir accompagné, m'avoir fait grandir techniquement et humainement, pour leur temps passé à m'aider, un grand merci à Jérôme, Fabrice, Mokrane, Faouzi, Nabil, Rachid, Mohamed, Benjamin, Carole, Bertrand, Nadia et Damien.

Merci également à Philippe qui m'a fait passer les entretiens techniques d'embauche et m'a invité à rejoindre son équipe en connaissance de cause et à Luc qui m'a fait confiance au point de m'embaucher avant que je n'obtienne le titre RNCP.

Merci enfin à tous ces auteurs qui ont bercé mes soirées et hanté mes nuits : Joshua Bloch, Bertrand Meyer, Andrew Hunt et David Thomas, Eric Freeman et Elisabeth Robson, Kathy Sierra et Bert Bates.

## BIBLIOGRAPHIE

Katy Sierra, Bert Bates (2008). **Head First Java**

Bertrand Meyer (2008). **Conception et programmation orientée objet**

Joshua Bloch (2017). **Effective Java (Third Edition)**

Elisabeth Robson, Eric Freeman (2004). **Head First Design Patterns**

Andrew Hunt, David Thomas (2005). **The Pragmatic Programmer: From Journeyman to Master**

## RESSOURCES

OWASP : <https://owasp.org/>

W3schools : <https://www.w3schools.com/>

Spring documentation : <https://spring.io/>

Angular documentation : <https://angular.io/>

Java 8 documentation : <https://docs.oracle.com/javase/8/docs/api/>

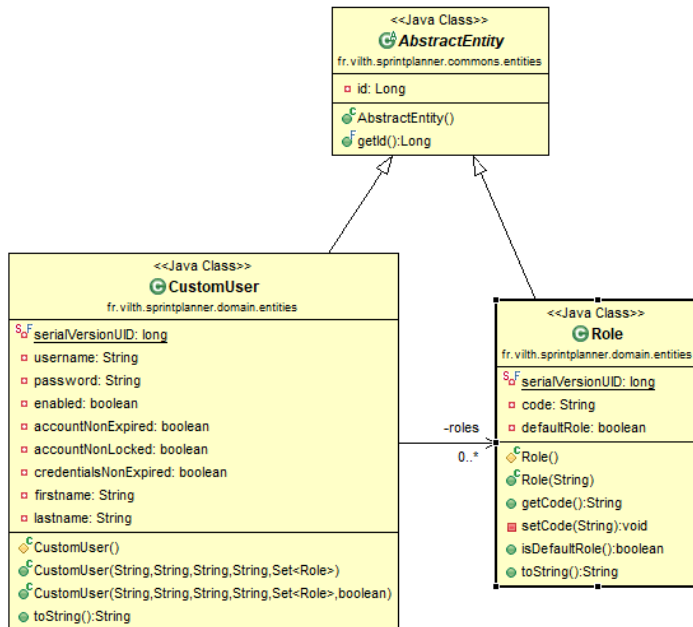
Et bien sûr, **stackoverflow** !

# ANNEXES

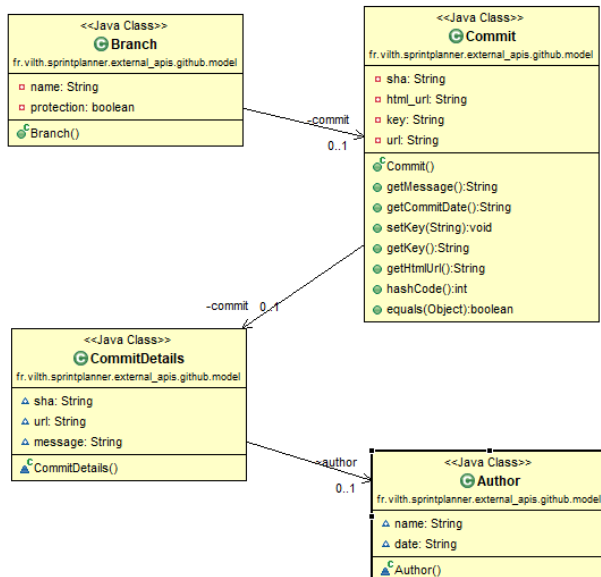
## ANNEXE 1 - Diagrammes de classe

Seul le diagramme de classe du domaine est représenté dans le dossier projet. Vous trouverez ci-dessous les diagrammes des autres composantes de l'application :

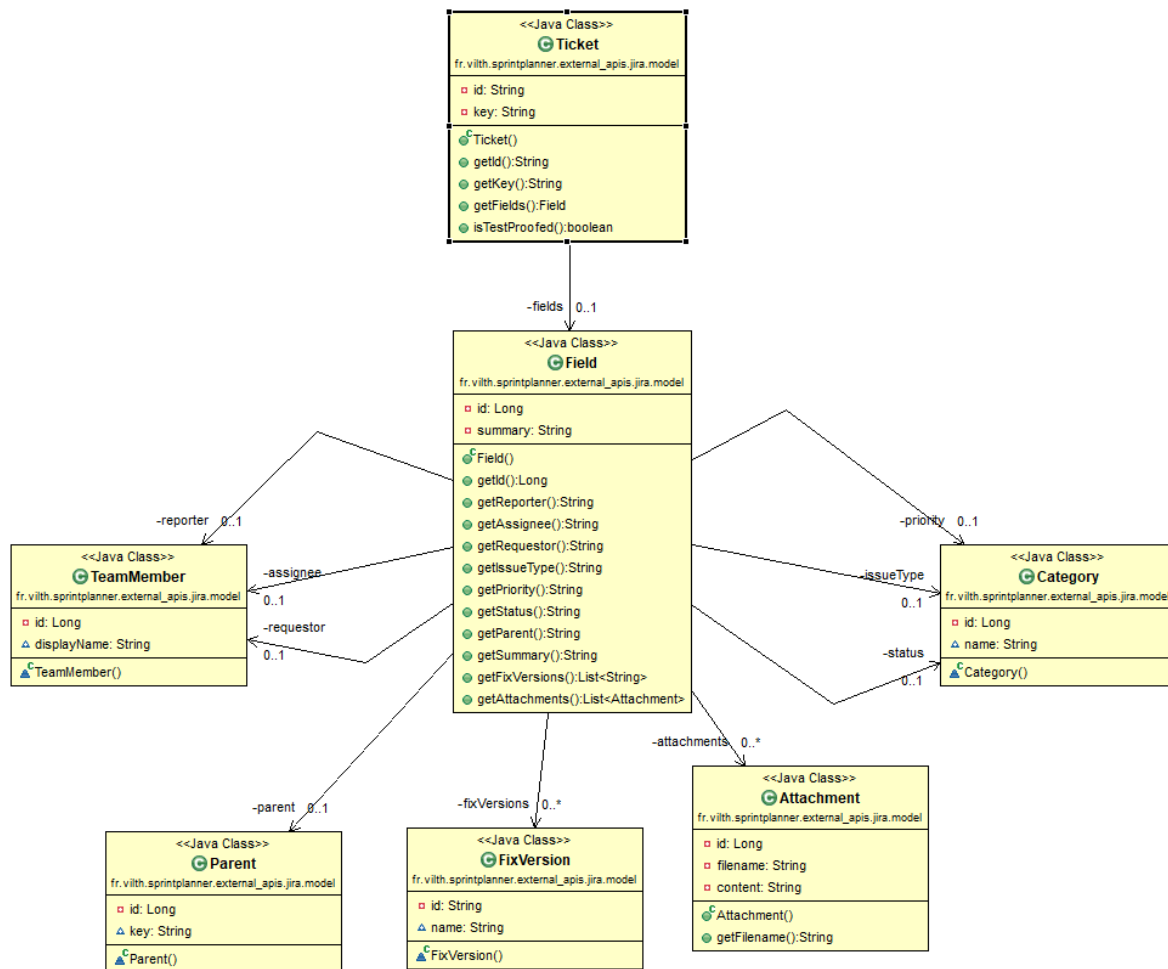
### Authentification



### Github Api



## Jira API



## ANNEXE 2 - DDL et DML

### Script de création de schéma

H2

```
-- create schema
CREATE SCHEMA IF NOT EXISTS `sprintplanner` AUTHORIZATION root;
USE `sprintplanner`;

-----
-- Table `sprintplanner`.`members`
-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`members` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(110) NOT NULL,
  `firstname` VARCHAR(50) NOT NULL,
  `lastname` VARCHAR(50) NOT NULL,
  `shift` ENUM('PAR', 'BGL') NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE `members_email_UNIQUE` (`email` ASC));

-----
-- Table `sprintplanner`.`tasks`
-- Table `sprintplanner`.`tasks`
-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`tasks` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `description` VARCHAR(255) NULL DEFAULT NULL,
  `email` VARCHAR(100) NOT NULL,
  `name` VARCHAR(10) NOT NULL,
  `manager_id` BIGINT(20) NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `tasks_member_id_FK`
    FOREIGN KEY (`manager_id`)
      REFERENCES `sprintplanner`.`members` (`id`));

-----
-- Table `sprintplanner`.`candidates`
-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`candidates` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `priority` INT(11) NOT NULL,
  `status` ENUM('AVAILABLE', 'UNAVAILABLE', 'CURRENT') NOT NULL,
  `member_id` BIGINT(20) NOT NULL,
  `task_id` BIGINT(20) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE `candidates_member_id_task_id_UNIQUE` (`member_id` ASC, `task_id` ASC),
  CONSTRAINT `candidates_member_id_FK`
    FOREIGN KEY (`member_id`)
      REFERENCES `sprintplanner`.`members` (`id`),
  CONSTRAINT `candidates_task_id_FK`
    FOREIGN KEY (`task_id`)
      REFERENCES `sprintplanner`.`tasks` (`id`));
```

```

]-----
-- Table `sprintplanner`.`projects`
]-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`projects` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(100) NOT NULL,
  `github_repo` VARCHAR(100) NOT NULL,
  `github_user` VARCHAR(100) NOT NULL,
  `name` VARCHAR(50) NOT NULL,
  `pi_duration` INT(11) NOT NULL,
  `sprint_duration` INT(11) NOT NULL,
  `trigram` VARCHAR(5) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE `projects_trigram_UNIQUE` (`trigram` ASC));

]-----
-- Table `sprintplanner`.`releases`
]-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`releases` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `pi` INT(11) NOT NULL,
  `releaser` VARCHAR(255) NULL DEFAULT NULL,
  `sprint` INT(11) NOT NULL,
  `version_number` VARCHAR(255) NULL DEFAULT NULL,
  `week` INT(11) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE `releases_pi_sprint_week_UNIQUE` (`pi` ASC, `sprint` ASC, `week` ASC));

]-----
-- Table `sprintplanner`.`roles`
]-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`roles` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `code` VARCHAR(256) NOT NULL,
  `default_role` VARCHAR(1) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE `roles_code_UNIQUE` (`code` ASC));

]-----
-- Table `sprintplanner`.`users`
]-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`users` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `account_non_expired` VARCHAR(1) NOT NULL,
  `account_non_locked` VARCHAR(1) NOT NULL,
  `credentials_non_expired` VARCHAR(1) NOT NULL,
  `enabled` VARCHAR(1) NOT NULL,
  `firstname` VARCHAR(256) NOT NULL,
  `lastname` VARCHAR(256) NOT NULL,
  `password` VARCHAR(256) NOT NULL,
  `username` VARCHAR(256) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE `users_username_UNIQUE` (`username` ASC));

]-----
-- Table `sprintplanner`.`user_role`
]-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`users_roles` (
  `user_id` BIGINT(20) NOT NULL,
  `role_id` BIGINT(20) NOT NULL,
  PRIMARY KEY (`user_id`, `role_id`),
  CONSTRAINT `users_roles_role_id_FK`
    FOREIGN KEY (`role_id`)
      REFERENCES `sprintplanner`.`roles` (`id`),
  CONSTRAINT `users_roles_user_id_FK`
    FOREIGN KEY (`user_id`)
      REFERENCES `sprintplanner`.`users` (`id`));

]-----
-- Table `sprintplanner`.`jobs`
]-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`jobs` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `title` VARCHAR(10) NOT NULL,
  `task` VARCHAR(10) NOT NULL,
  `active` VARCHAR(1) NOT NULL,
  PRIMARY KEY (`id`));

```

## MySQL

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- Schema sprintplanner
CREATE SCHEMA IF NOT EXISTS `sprintplanner`
DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci;
USE `sprintplanner` ;

-----
-- Table `sprintplanner`.`members`
-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`members` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(110) NOT NULL,
  `firstname` VARCHAR(50) NOT NULL,
  `lastname` VARCHAR(50) NOT NULL,
  `shift` ENUM('FAR', 'BGL') NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `members_email_UNIQUE` (`email` ASC) VISIBLE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-----
-- Table `sprintplanner`.`tasks`
-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`tasks` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `description` VARCHAR(255) NULL DEFAULT NULL,
  `email` VARCHAR(100) NOT NULL,
  `name` VARCHAR(10) NOT NULL,
  `manager_id` BIGINT(20) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `tasks_member_id_IDX` (`manager_id` ASC) VISIBLE,
  CONSTRAINT `tasks_member_id_FK`
    FOREIGN KEY (`manager_id`)
      REFERENCES `sprintplanner`.`members` (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-----
-- Table `sprintplanner`.`tasks`
-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`tasks` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `description` VARCHAR(255) NULL DEFAULT NULL,
  `email` VARCHAR(100) NOT NULL,
  `name` VARCHAR(10) NOT NULL,
  `manager_id` BIGINT(20) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `tasks_member_id_IDX` (`manager_id` ASC) VISIBLE,
  CONSTRAINT `tasks_member_id_FK`
    FOREIGN KEY (`manager_id`)
      REFERENCES `sprintplanner`.`members` (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-----
-- Table `sprintplanner`.`candidates`
-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`candidates` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `priority` INT(11) NOT NULL,
  `status` ENUM('AVAILABLE', 'UNAVAILABLE', 'CURRENT') NOT NULL,
  `member_id` BIGINT(20) NOT NULL,
  `task_id` BIGINT(20) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `candidates_member_id_task_id_UNIQUE` (`member_id` ASC, `task_id` ASC) VISIBLE,
  INDEX `candidates_member_id_IDX` (`member_id` ASC) VISIBLE,
  INDEX `candidates_task_id_IDX` (`task_id` ASC) VISIBLE,
  CONSTRAINT `candidates_member_id_FK`
    FOREIGN KEY (`member_id`)
      REFERENCES `sprintplanner`.`members` (`id`),
  CONSTRAINT `candidates_task_id_FK`
    FOREIGN KEY (`task_id`)
      REFERENCES `sprintplanner`.`tasks` (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```



```

-----
-- Table `sprintplanner`.`projects`
-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`projects` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(100) NOT NULL,
  `github_repo` VARCHAR(100) NOT NULL,
  `github_user` VARCHAR(100) NOT NULL,
  `name` VARCHAR(50) NOT NULL,
  `pi_duration` INT(11) NOT NULL,
  `sprint_duration` INT(11) NOT NULL,
  `trigram` VARCHAR(5) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `projects_trigram_UNIQUE` (`trigram` ASC) VISIBLE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-----
-- Table `sprintplanner`.`releases`
-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`releases` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `pi` INT(11) NOT NULL,
  `releaser` VARCHAR(255) NULL DEFAULT NULL,
  `sprint` INT(11) NOT NULL,
  `version_number` VARCHAR(255) NULL DEFAULT NULL,
  `week` INT(11) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `releases_pi_sprint_week_UNIQUE` (`pi` ASC, `sprint` ASC, `week` ASC) VISIBLE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE TABLE IF NOT EXISTS `sprintplanner`.`roles` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `code` VARCHAR(256) NOT NULL,
  `default_role` VARCHAR(1) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `roles_code_UNIQUE` (`code` ASC) VISIBLE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-----
-- Table `sprintplanner`.`users`
-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`users` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `account_non_expired` VARCHAR(1) NOT NULL,
  `account_non_locked` VARCHAR(1) NOT NULL,
  `credentials_non_expired` VARCHAR(1) NOT NULL,
  `enabled` VARCHAR(1) NOT NULL,
  `firstname` VARCHAR(256) NOT NULL,
  `lastname` VARCHAR(256) NOT NULL,
  `password` VARCHAR(256) NOT NULL,
  `username` VARCHAR(256) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `users_username_UNIQUE` (`username` ASC) VISIBLE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-----
-- Table `sprintplanner`.`user_role`
-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`users_roles` (
  `user_id` BIGINT(20) NOT NULL,
  `role_id` BIGINT(20) NOT NULL,
  PRIMARY KEY (`user_id`, `role_id`),
  INDEX `users_roles_user_id_IDX` (`user_id` ASC) VISIBLE,
  INDEX `users_roles_role_id_IDX` (`role_id` ASC) VISIBLE,
  CONSTRAINT `users_roles_role_id_FK`
    FOREIGN KEY (`role_id`)
      REFERENCES `sprintplanner`.`roles` (`id`),
  CONSTRAINT `users_roles_user_id_FK`
    FOREIGN KEY (`user_id`)
      REFERENCES `sprintplanner`.`users` (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-----
-- Table `sprintplanner`.`jobs`
-----
CREATE TABLE IF NOT EXISTS `sprintplanner`.`jobs` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `title` VARCHAR(10) NOT NULL,
  `task` VARCHAR(10) NOT NULL,
  `active` VARCHAR(1) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

```

## Script d'insertion de données

```
USE `sprintplanner`;

insert into projects(name, email, github_user, github_repo, pi_duration, sprint_duration, trigram) values
('sprintplanner', 'sprintplanner@bot', 'vilth83', 'sprint-planner', 3, 3, 'SPL');
COMMIT;

insert into members (email, firstname, lastname, shift) values
('frank@marshall', 'Frank', 'Marshall', 'BGL'),
('thierry@villepreux', 'Thierry', 'Villepreux', 'BGL'),
('ioana@ardelean', 'Ioana', 'Ardelean', 'BGL'),
('tantely@andria', 'Tantely', 'Andria', 'BGL'),
('nathalie@robert', 'Nathalie', 'Robert', 'BGL'),
('jonathan@luminuku', 'Jonathan', 'Luminuku', 'BGL'),
('david@dik', 'David', 'Dik', 'BGL'),
('remy@guilloux', 'Remy', 'Guilloux', 'PAR'),
('annes@jehu', 'Anne-Sophie', 'Jehu', 'PAR'),
('philippe@amice', 'Philippe', 'Amice', 'PAR'),
('louis@godlewski', 'Louis', 'Godlewski', 'PAR'),
('patricia@leclerc', 'Patricia', 'Leclerc', 'PAR'),
('lea@limelette', 'Lea', 'Limelette', 'PAR');
COMMIT;

insert into releases (pi, sprint, week, version_number, releaser)
values (1,1,0,'v0', 'start'),
(1,1,1,'v1.1.1', 'bob'),
(1,1,2,'v1.1.2', 'louis'),
(1,1,3,'v1.1.3', 'lea'),
(1,2,1,'v1.2.1', 'frank');
COMMIT;

insert into tasks (description, email, name, manager_id) values
('manage releases', 'releaser@mail', 'releaser', 1),
('manage support', 'support@mail', 'technical', 1),
('handle tests', 'test@mail', 'tester', 1),
('manage support', 'support@mail', 'functional', 1);
COMMIT;

insert into candidates (priority, `status`, member_id, task_id) values
(2, 'CURRENT', 1, 1),
(1, 'AVAILABLE', 2, 1),
(0, 'AVAILABLE', 3, 1),
(1, 'CURRENT', 4, 2),
(0, 'AVAILABLE', 5, 2),
(1, 'CURRENT', 6, 2),
(0, 'AVAILABLE', 7, 2),
(0, 'AVAILABLE', 8, 3),
(1, 'CURRENT', 9, 3),
(1, 'CURRENT', 10, 4),
(0, 'AVAILABLE', 11, 4),
(1, 'CURRENT', 12, 4),
(0, 'AVAILABLE', 13, 4);
COMMIT;

INSERT INTO `users` (id, account_non_expired, firstname, lastname, password, username, account_non_locked, credentials_non_expired, enabled)
VALUES
(1, 'T', 'Thierry', 'VILLEPREUX', '$2a$10$CpVc.yvqf5aoxNH9bz.IeU67PEm2WgLl8YrUOi2D/VYTW/MTp9aS', 'admin', 'T', 'T', 'T'),
(2, 'T', 'Thierry', 'VILLEPREUX', '$2a$10$CpVc.yvqf5aoxNH9bz.IeU67PEm2WgLl8YrUOi2D/VYTW/MTp9aS', 'user', 'T', 'T', 'T');
COMMIT;

INSERT INTO `roles` VALUES
(1, 'ROLE_USER', 'T'),
(2, 'ROLE_ADMIN', 'F');
COMMIT;

INSERT INTO `users_roles`
VALUES (1,2), (2,1);
COMMIT;

INSERT INTO `jobs` VALUES
(1, 'mail', 'releaser', 'F'),
(2, 'mail', 'support', 'F'),
(3, 'mail', 'tester', 'F'),
(4, 'roundRobin', 'releaser', 'F'),
(5, 'roundRobin', 'support', 'F'),
(6, 'roundRobin', 'tester', 'F');
COMMIT;
```

## ANNEXE 3 – RESULTATS DE RECHERCHE

### Exemple d'implémentation trouvée pour l'envoi d'un mail sur StackOverflow

```
private static MimeMessage createEmail(String to, String cc, String from,
    String subject, String bodyText) throws MessagingException {
    Properties props = new Properties();
    Session session = Session.getDefaultInstance(props, null);
    MimeMessage email = new MimeMessage(session);
    InternetAddress tAddress = new InternetAddress(to);
    InternetAddress cAddress = cc.isEmpty() ? null
        : new InternetAddress(cc);
    InternetAddress fAddress = new InternetAddress(from);
    email.setFrom(fAddress);
    if (cAddress != null) {
        email.addRecipient(javax.mail.Message.RecipientType.CC, cAddress);
    }
    email.addRecipient(javax.mail.Message.RecipientType.TO, tAddress);
    email.setSubject(subject);
    email.setText(bodyText);
    return email;
}

private static Message createMessageWithEmail(MimeMessage email)
    throws MessagingException, IOException {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    email.writeTo(baos);
    String encodedEmail = Base64
        .encodeBase64URLSafeString(baos.toByteArray());
    Message message = new Message();
    message.setRaw(encodedEmail);
    return message;
}

public static void Send(Gmail service, String recipientEmail,
    String ccEmail, String fromEmail, String title, String message)
    throws IOException, MessagingException {
    Message m = createMessageWithEmail(createEmail(recipientEmail, ccEmail,
        fromEmail, title, message));
    service.users().messages().send("me", m).execute();
}
```

### Implémentation finale

#### Implémentation du service d'envoi de mail

```
@Override
public void sendMail(Mail mail) throws MessagingException {
    MimeMessage message = javaMailSender.createMimeMessage();
    MimeMessageHelper helper = new MimeMessageHelper(message, true,
        ENCODING);
    helper.setFrom(mail.getSender());
    helper.setTo(mail.getRecipients().toArray(new String[0]));
    helper.setSubject(mail.getSubject());
    message.setContent(mail.getContent(), TYPE);
    javaMailSender.send(message);
}
```

#### Implémentation de la configuration de JavamailSender

```
@Bean
@ConfigurationProperties(prefix = "spring.mail")
public JavaMailSender getJavaMailSender() {
    final JavaMailSenderImpl javaMailSender = new JavaMailSenderImpl();
    javaMailSender.setHost(host);
    javaMailSender.setProtocol(protocol);
    javaMailSender.setPort(port);
    final Properties props = javaMailSender.getJavaMailProperties();
    props.put("mail.smtp.auth", FALSE);
    props.put("mail.smtp.starttls", FALSE);
    props.put("mail.debug", FALSE);
    return javaMailSender;
}
```