

# Projet Java : Création d'un CRM pour le S.H.I.E.L.D.



Marie-Cécile PEYRAN

Promotion :

Alexandre CLAIN Pilote de  
formation

Sébastien XAVIER CARLOS

Développeur informatique 2019

Julien ROMEJON

Gabriel DESSERÉ-SUSINI

Accompagnateur du projet

# Table des matières

Table des matières .....	2
1 Présentation du projet .....	2
1.1 Le contexte .....	2
2 Gestion de projet.....	2
2.1 Outils de gestion de projet.....	2
2.2 Répartition des tâches.....	4
2.3 Déroulement du projet .....	5
3 Définition du besoin .....	6
3.1 Analyse du besoin.....	6
3.2 Personas .....	6
3.3 Cartographie fonctionnelle .....	8
3.3.1 Présentation des modules thématiques .....	8
3.3.2 Schéma et priorisation des développements.....	10
4 La modélisation UML.....	10
4.1 Analyse de la mission .....	10
4.2 Outils utilisés .....	11
4.3 Les différents diagrammes .....	13
4.3.1 Diagramme de cas d'utilisation macro.....	13
4.3.2 Diagramme de cas d'utilisation micro.....	14
4.3.3 Diagramme de flux du cycle de vie d'une donnée .....	14
4.3.4 Diagramme de classe.....	17
5 La conception .....	18
5.1 Analyse de la mission .....	18
5.2 Définition du design .....	18
5.2.1 Outils utilisés .....	18
5.2.2 Définition de la charte graphique .....	18
5.2.3 Réalisation de la maquette .....	20
5.3 Conception et mise en place de la base de données .....	22
5.3.1 Outils utilisés .....	22
5.3.2 MCD / MLD .....	22
5.3.3 Création & sécurisation.....	24
5.4 Architecture logicielle.....	24
5.4.1 Présentation de l'interface de programmation d'application .....	24
5.4.2 Définition de l'architecture .....	26
6 Réalisation technique .....	29

6.1	Analyse de la mission .....	29
6.2	Outils utilisés .....	29
6.2.1	Les langages.....	29
6.2.2	En ce qui concerne le choix de l'IDE : .....	29
6.2.3	Les Framework .....	29
6.2.4	Les librairies.....	30
6.2.5	Outils Annexes.....	30
6.3	Versionnage.....	31
6.4	Règles de standardisation .....	32
6.5	Infrastructure .....	33
6.5.1	Packages fonctionnels .....	33
6.5.2	Sous-packages thématiques.....	34
6.6	Code.....	35
6.6.1	Fichier de configuration .....	35
6.6.2	Présentation du CRUD (incident) .....	35
6.6.3	Présentation des Token.....	38
6.6.4	Présentation des Mapper.....	39
6.6.5	Présentation du front-end.....	41
6.7	Tests & recettage .....	43
6.7.1	Fiche de tests.....	43
7	Manuel d'utilisation .....	45
8	Évolutions à venir .....	46
9	Conclusion .....	47
9.1	Rappel du contexte .....	47
9.2	Conformité des réalisations .....	47
9.3	Problématiques rencontrées.....	48
9.4	Bonnes pratiques et axes d'améliorations .....	49
9.5	Bilan individuel .....	50
9.6	Bilan global .....	51
9.7	Remerciements .....	51
9.8	Annexes .....	52
9.8.1	Première planification .....	52
8.8.2	Personas .....	54

# 1 Présentation du projet

## 1.1 Le contexte

Le S.H.I.E.L.D. souhaite mettre en place un système de gestion de relation client (CRM ou Customer Relationship Management). Le projet se déroule dans une ère qui se situe post accords de Sokovie. L'intention du S.H.I.E.L.D. de connaître tous les citoyens ainsi que leurs possibles multiples identités semble justifiée.

Les objectifs principaux sont de mesurer l'efficacité des actions des Avengers et évaluer le niveau de satisfaction de la société civile.

Cet outil aura également une importante fonction de surveillance puisqu'il facilitera le signalement au S.H.I.E.L.D. d'évènements nécessitant l'intervention d'un super-héros. Il permettra également de constituer une base de données de référence regroupant des informations sur les parties civiles, les super-héros et leurs antagonistes malveillants. Une attention toute particulière sera donc accordée à l'outil de base de données afin de garantir sa robustesse et son niveau de sécurisation.

La multiplicité des supports et des cas d'utilisations (du civil appelant au secours à l'agent administratif du S.H.I.E.L.D. en passant par le super-héros en intervention) nécessite une grande souplesse dans les modalités de consultation. Ainsi, une application hybride, consultable aussi bien sur supports mobiles qu'en version web paraît tout indiquée.

Un manuel d'utilisation est attendu pour faciliter l'accompagnement au changement des utilisateurs.

Par ailleurs, en tant que service informatique du S.H.I.E.L.D. nous respecterons les standards du service pour la production de logiciel interne.

- Le langage de développement Java est un prérequis.

Dans ce projet nous faisons face à une contrainte spécifique liée au temps. En effet nous disposons d'un peu plus de deux mois pour effectuer l'analyse et la conception du projet mais de seulement deux semaines et demie pour réaliser le code. C'est un challenge que nous relevons cependant.

# 2 Gestion de projet

## 2.1 Outils de gestion de projet

Afin de réaliser ce projet, nous avons choisi d'utiliser différents outils nous permettant d'avoir une organisation claire et encadrée.

Concernant la gestion des tâches, nous nous sommes tournés vers l'outil « *Trello* ». Outil de collaboration et de gestion sous forme de tickets. Afin d'avoir des objectifs précis nous avons réparti notre Trello en cinq parties principales :

- À faire
- En cours
- À rédiger
- À valider
- Terminé

Au sein de chaque partie nous avons donc créé les tâches à réaliser. Ces dernières reflètent les étapes de notre projet. Un ou plusieurs membres de l'équipe sont ensuite attribués aux tâches créées. Il est aussi possible d'ajouter des deadlines aux tâches afin de rester dans les délais de réalisation des modules du projet.

Nous avons aussi créé trois parties annexes découpées comme suit :

- **Questions :**

Regroupe toutes les questions relatives au projet à demander à l'accompagnateur du projet ainsi qu'au pilote de notre formation

- **Ressources**

Regroupe les fichiers importants du projet sur lesquels nous devons nous appuyer. Les résultats de recherches ont aussi été regroupés à l'intérieur de cette partie.

e.g. [La grille d'évaluation du projet](#)

- **Identifiants**

Regroupe les identifiants et liens permettant l'accès aux informations partagées.

e.g. [Liens onedrive](#)

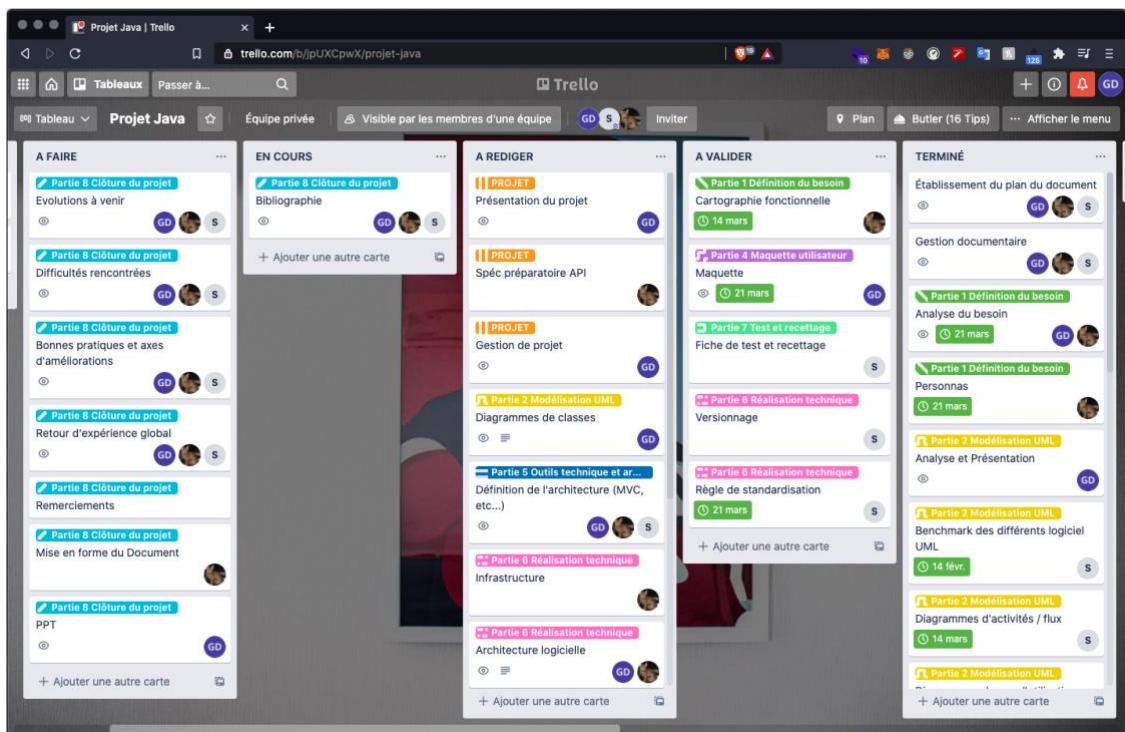


Figure 1 : Aperçu du Trello du projet

Concernant l'arborescence du projet et les éléments composant ce dernier nous avons choisi « [OneDrive](#) ». Il s'agit d'un outil de partage et de collaboration de la suite Microsoft Office qui permet de stocker des fichiers dans le cloud et d'avoir un accès à ces derniers en lecture et écriture sur une multitude d'appareils. Il est tout à fait possible de travailler à plusieurs sur un même document au même moment.

OneDrive nous permet de suivre les modifications apportées par toutes les personnes ayant les accès, de la même manière que sur Git, nous pouvons ainsi voir et contrôler les modifications ; nous pouvons même les annuler. Cependant il faudra être prudent lors de la modification d'une même section par deux utilisateurs en même temps. En effet la deuxième personne souhaitant réaliser la synchronisation du document modifié se retrouvera avec des conflits à résoudre avant toute choses.

Nom	Modifié	Modifié ...	Taille du ...	Partage
Archivage	17 décembre 2...	DESSERE SUSI...	0 éléments	Partagé
Organisation	17 décembre 2...	DESSERE SUSI...	3 éléments	Partagé
Réalisations	17 décembre 2...	DESSERE SUSI...	10 éléments	Privé
Ressources	17 décembre 2...	DESSERE SUSI...	8 éléments	Partagé
Soutenance	17 décembre 2...	DESSERE SUSI...	0 éléments	Partagé

Figure 2 : Aperçu du OneDrive du projet

À propos de la communication, nous avons utilisé l'application de messagerie instantanée « [WhatsApp](#) » pour communiquer hors réunion. Toujours faisant partie de la suite Microsoft Office l'application « [Microsoft Teams](#) » nous servi de moyen de communication direct lors des réunions.



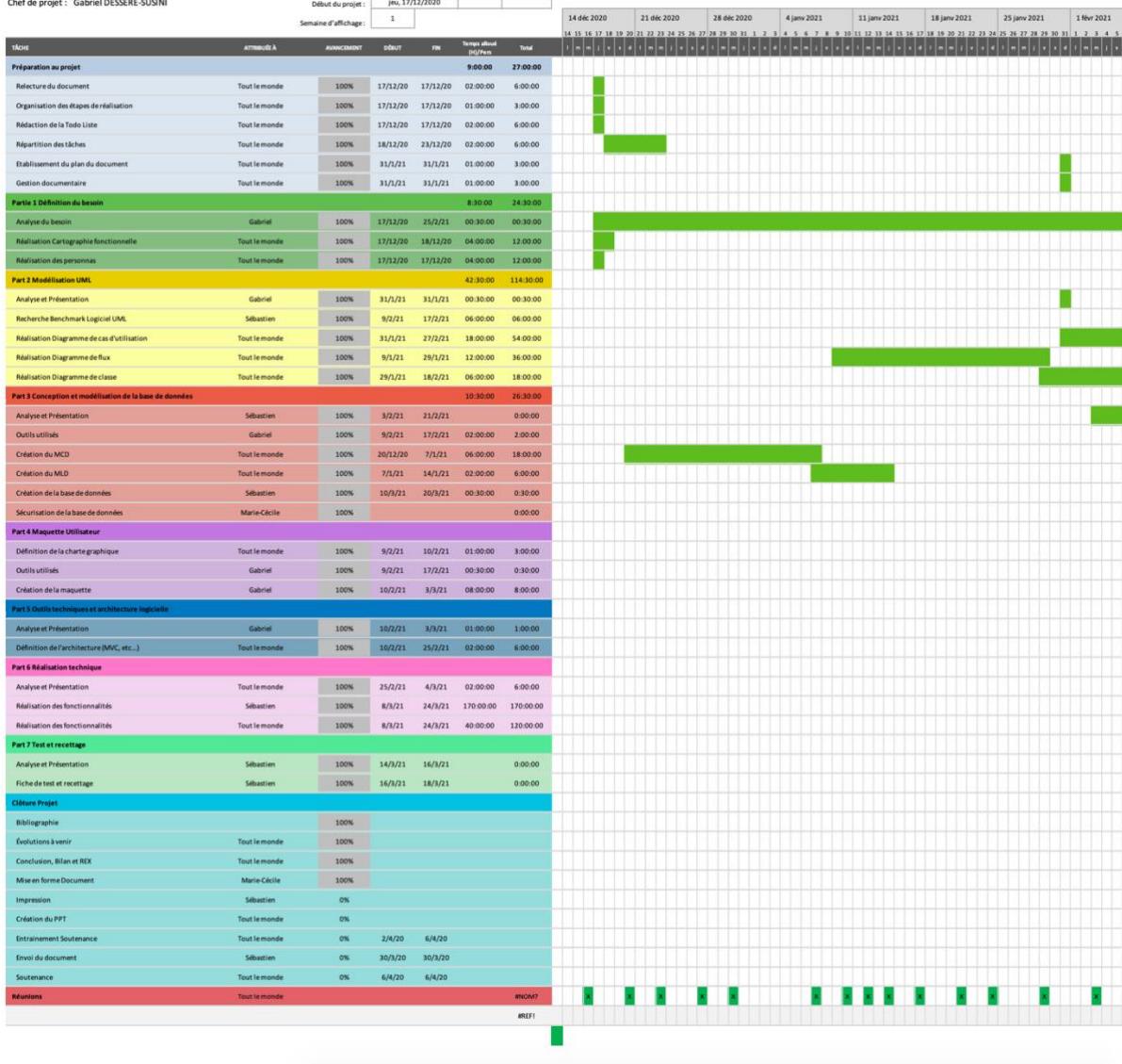
## 2.2 Répartition des tâches

Dans une optique d'organisation globale ainsi que pour la répartition générale des tâches nous avons choisi d'utiliser un « [Diagramme de Gantt](#) » (que vous pourrez retrouver [ici](#)), outil permettant de visualiser dans le temps les diverses tâches composant un projet et leur état d'avancement. Il est donc composé de plusieurs parties :

- Une liste de tâches à envisager
- L'attribution de ces tâches
- Leur date de début et de fin
- Le temps de réalisation global de la tâche et individuel
- L'avancement de la complétion de la tâche
- Le chevauchement éventuel des tâches entre elles

## Projet Infrastructure

S.H.I.E.L.D. R&D  
Chef de projet : Gabriel DESSEUR-SUSINI



## 2.3 Déroulement du projet

Pour la mise en place des réunions, nous les avons prévues en fonction du déroulement du projet. Nous nous retrouvions pour une réunion deux fois par semaine, tous les mercredi et dimanche soir, lors de la phase d'analyse du projet qui dura environ deux mois et demi. Lors de la phase de la réalisation technique, d'une durée de deux semaines et demie environ, nous nous retrouvions jusqu'à cinq fois par semaine. Dès la clôture d'une partie du projet, une réunion était organisée afin de valider collectivement les productions de chacun. De plus, chaque réalisation devait être expliquée aux membres ne les ayant pas réalisés afin qu'une compréhension globale du projet soit acquise par tous. Afin de pouvoir partager nos connaissances et pouvoir approfondir nos réflexions dans la façon de coder, nous avons choisi d'utiliser les méthodes de pair programming qui nous ont paru très productives.

La répartition des rôles principaux de ce projet fut la suivante :

Marie-Cécile : Rédactrice

Sébastien : Chargé de communication

Gabriel : Chef de projet

## 3 Définition du besoin

### 3.1 Analyse du besoin

Avant toute étude technique, notre premier objectif est de prendre connaissance de la réalité métier de nos clients. Une fois que nous comprendrons leur fonctionnement actuel, nous pourrons envisager ensemble comment prendre en compte les besoins dans la future application, voir même proposer des fonctionnalités supplémentaires qui viendront améliorer leur quotidien.

Après lecture du cahier des charges du S.H.I.E.L.D., nous avons donc décidé de procéder en deux temps.

Tout d'abord nous approfondirons notre connaissance des profils de nos clients et de leur environnement de travail à travers la réalisation de personas.

Enfin, en partant de cette vision d'ensemble nous synthétiserons l'attendu en cartographiant les grandes fonctionnalités nécessaires.

### 3.2 Personas

Les personas font partie de la panoplie d'outils entourant la rédaction des user stories. Il s'agit d'imaginer des personnages archétypaux qui représenteront le groupe d'utilisateur auquel ils appartiennent.

La réflexion s'articule autour de trois axes principaux :

- **Biographie rapide du persona** : lui inventer un cadre de vie permet de poser un contexte métier concret et réaliste.
- **But/Comportement** : on détermine ici les attitudes et aspirations du persona. En s'appuyant sur la biographie, des besoins jusqu'à lors non identifiés peuvent apparaître au cours de cette exercice.
- **Qu'est-ce qu'on veut lui apporter** : Synthèse des besoins du persona identifiés à partir des sections précédentes.

De par sa mise en situation, la réalisation de personas nous a paru l'outil le plus adapté, nous permettant notamment de simuler les entretiens métiers que nous ne pouvions réaliser.

Si nous regroupons les différents utilisateurs de l'application, trois grands types de profils émergent :

- **Le Civil** : nous cherchons ici à imaginer les utilisations de l'application que pourrait avoir le citoyen lambda.
- **L'Employé** : affilié au S.H.I.E.L.D., il travaille dans l'ombre des Super-Héros. Sans lui rien ne serait possible. La connaissance de sa réalité métier est un élément clé de la réussite de notre projet.
- **Le Super-Héros** : nous nous interrogeons alors sur le quotidien professionnel du super-héros. De quoi a-t-il besoin et que lui manque-t-il ?

Dans le vocabulaire des user stories, l'« abuser » désigne un utilisateur qui viendrait détourner l'usage premier de l'application à des fins malhonnêtes. Sa prise en compte, dès l'analyse du besoin, permet de recentrer les questions de sécurité au cœur de la conception de l'application.

Bien qu'il ne puisse être considéré comme un utilisateur à part entière, il reste donc un profil que nous ne devons pas ignorer dans notre réflexion. Il s'agit du **Super-Vilain** dont le but est de nuire au S.H.I.E.L.D. et à l'équipe des Avengers.

### Voir les personas du Super-Vilain et du Civil en annexe

Sources d'inspiration pour la réalisation des personas :

<https://blog.myagilepartner.fr/index.php/2017/07/24/bien-ecrire-son-persona-dans-un-projet-agile/>

<https://blog.myagilepartner.fr/index.php/2020/06/09/feature-user-story-epic/>

## Hulk [Dr Bruce Banner], Super-héros



### CONTEXTE

Un peu soupe au lait, il a tendance à se laisser emporter, ce qui provoque des dégâts collatéraux.  
Impulsif, il intervient sans prévenir personne.

Solitaire

Fait partie des Avengers et intervient avec eux sur les missions.

### BUT / COMPORTEMENTS :

Est l'objet de nombreuses crises (destructions de biens) et litiges (procès)

Effectue souvent sa mission sans déclaration d'incident ce qui a pour conséquence de générer des incidents côté civil ou nécessite la création après coup d'une déclaration d'incident.

Peut déclarer des incidents même s'il ne le fait pas souvent et peut demander des renforts quand la situation se complique.

Reçoit des demandes d'interventions.

Son taux de génération d'incident fait qu'il est sélectionné en dernier recours parmi les super-héros disponibles, particulièrement si une grande force de frappe est nécessaire.

Reçoit les courriels de ses fans. Et anti-fans mais ceux-là ont tendance à l'énerver.

### QU'EST-CE QU'ON VEUT LUI APPORTER ?

Des alertes : être informé le plus rapidement possible d'une nouveauté

Accès aux informations concernant son périmètre d'activité

Info sur ces indicateurs.

Affordance des fonctionnalités : l'UX-UI a une grande importance.

### BUT / COMPORTEMENTS :

Reçoit les résultats du sondage et/ou de l'enquête terrain de l'incident pris en charge.

Peut rédiger des rapports mais comme il n'a pas le temps, il fait souvent appel à une personne tierce

Figure 3 : Personas de Dr. Bruce Banner

Joyce Flores, Membre du S.H.I.E.L.D.	
	<b>CONTEXTE</b> Ouverte d'esprit, elle est habituée à devoir gérer des situations peu ordinaires.  Secrétaire principale salariée du S.H.I.E.L.D., elle s'occupe de l'administratif pour l'équipe des Avengers et encadre l'équipe administrative.
<b>BUT / COMPORTEMENTS :</b> Création d'incidents Rédaction de rapports (création de Super-Vilain) Rédaction de litiges Rédaction de crises Mise à jour de toutes les bases de données de Civils, Super-Héros....	<b>QU'EST-CE QU'ON VEUT LUI APPORTER ?</b> Affordance des fonctionnalités : l'UX-UI a une grande importance. Des interfaces pour gérer les différentes saisies à effectuer. Des reportings et statistiques généraux.

Figure 4 : Personas de Joyce Flores Agent du S.H.I.E.L.D.

### 3.3 Cartographie fonctionnelle

#### 3.3.1 Présentation des modules thématiques

Suite à l'analyse du besoin que nous venons d'exécuter, nous allons pouvoir réaliser une « cartographie fonctionnelle » de notre future application.

Nous avons identifié les différents blocs de fonctionnalités nécessaires à l'accomplissement de l'application :

##### « Administration du système d'information »

Il nous a paru important de faire apparaître un bloc supplémentaire aux modules demandés qui nous permettra de pouvoir gérer les droits des différents utilisateurs. Nous établirons les règles de fonctionnement de l'application (tout élément concernant le noyau de l'application : connexion à la base de données, accès à distance...).

##### « Gestion des données civils »

Regroupe l'ensemble des fonctionnalités concernant l'identité des civils « lambda ». Le civil peut déclarer des incidents, des litiges....

##### « Gestion des données organisations »

Regroupe l'ensemble des fonctionnalités concernant les entités « Organisations » auxquelles peuvent être rattachés des civils.

Les organisations peuvent elles-aussi déclarer des incidents et des litiges. Elles peuvent également intervenir sur certaines missions.

##### « Gestion des données super-héros »

Regroupe l'ensemble des fonctionnalités concernant les civils de type super-héros. Ils peuvent déclarer des incidents et des litiges. Ils peuvent bien sûr intervenir en mission. Par la suite, ils doivent rédiger un rapport ou déléguer cette tâche à un agent agréé.

A noter : certains super-héros n'ayant pas signés les accords de Sokovia, ils ne sont pas identifiés civilement et ne peuvent donc intervenir sur certains territoires. S'ils interviennent dans ces zones d'action, ils sont à ce moment-là considérés comme hors-la-loi et déclenchent une crise.

##### « Gestion des données super-vilains »

Regroupe l'ensemble des fonctionnalités concernant les civils de type super-vilains. Le S.H.I.E.L.D. cherche à les identifier et les localiser. Les super-vilains peuvent être la cause d'incidents et donc apparaître dans les rapports de missions.

Par définition, le super-vilain cherche à nuire. Il peut tenter de voler ou de corrompre des données ainsi que de perturber les interventions des Avengers en signalant de faux incidents. Il peut également envoyer des lettres de menace. Pour cela, il revêt l'identité d'un citoyen lambda.

Ajoutons à cela, que la découverte d'un nouveau super-vilain engendre une crise.

#### **« Déclarations d'incidents »**

Regroupe l'ensemble des fonctionnalités concernant les déclarations d'incidents.

Tous les civils peuvent déclarer un incident. Le niveau de fiabilité du civil détermine le processus de validation de la demande d'intervention. Une alerte est alors envoyée aux intervenants concernés. *Exemple : si l'incident est déclaré par un super-héros ou un employé du S.H.I.E.L.D., il sera automatiquement validé.*

A noter : parfois les super-héros interviennent dans l'urgence. La déclaration d'incident est alors rédigée à posteriori.

#### **« Gestion des missions »**

Regroupe l'ensemble des fonctionnalités relatives à la gestion des missions.

La validation d'une déclaration d'incident engendre une demande d'intervention. Les héros les plus à même de résoudre cet incident seront affectés à la mission. Ils sont avertis via le système d'alerte.

Une fois la mission menée à bien, un rapport de mission doit être réalisé (*cf. super-héros*). Dans tous les cas, le super-héros concerné devra valider le rapport de mission.

#### **« Gestion des feedbacks »**

Regroupe l'ensemble des fonctionnalités relatives aux feedbacks.

A chaque fin de mission, un sondage est réalisé auprès de la population. Lorsque le niveau de satisfaction requis n'est pas atteint, une alerte est envoyée au personnel concerné (agents agréés ou super-héros).

Le courrier réceptionné est classé manuellement par un agent : courrier positif ou courrier négatif. Dans le futur, un système de classification par intelligence artificielle sera implanté.

Un système de gestion des feedbacks des réseaux sociaux sera mis en place.

#### **« Gestion des litiges »**

Regroupe l'ensemble des fonctionnalités correspondantes aux litiges.

Un litige peut être déclaré par des entités de type civiles. Sa déclaration génère une alerte aux intervenants concernés. Ces derniers qualifient le litige selon son niveau de gravité. Ils vont ensuite réaliser une enquête de terrain et alimenter la fiche litige au fil de l'eau.

Une fois tous les éléments rassemblés pour la résolution du litige, l'agent le clôture.

#### **Synthèse et reporting :**

Une fois l'application éprouvée par les utilisateurs et qu'ils auront réalisé leur feedback sur l'application, ils seront plus à même de définir leurs besoins en données et outils d'aide à la prise de décision. Nous laissons cette partie pour un second temps.

### 3.3.2 Schéma et priorisation des développements

Voici la cartographie des grandes fonctionnalités :  
*Une livraison pour le mois d'avril est prévue pour les éléments en vert.*

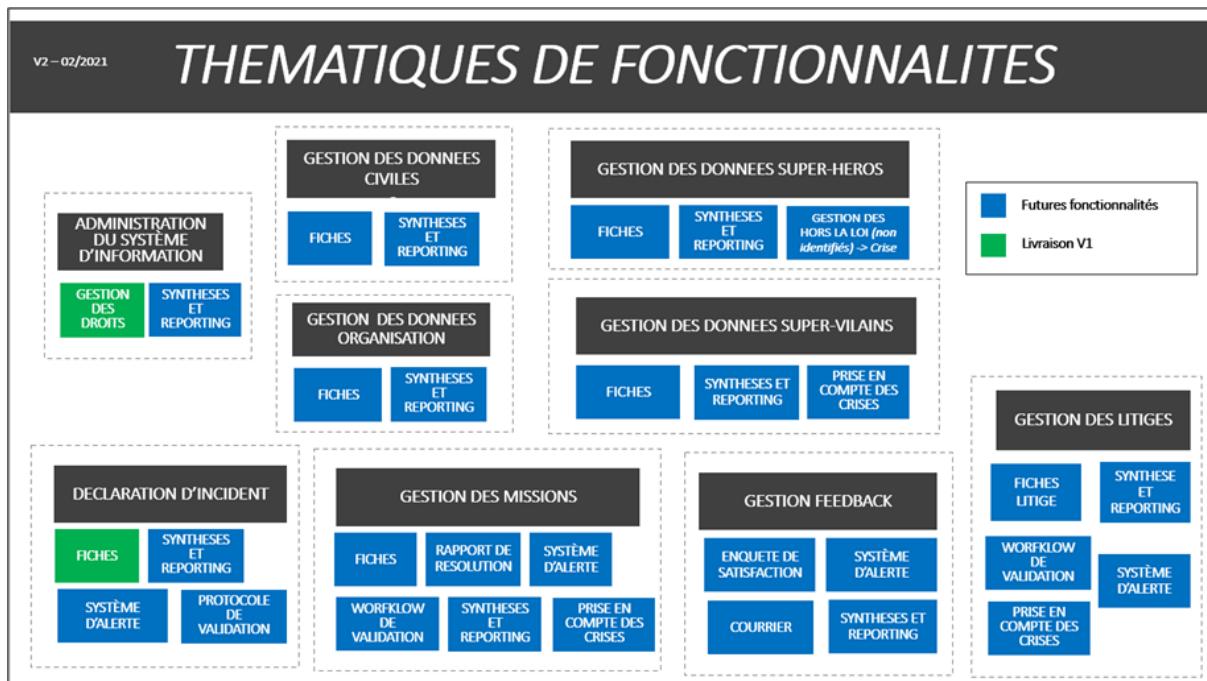


Figure 5 : Cartographie fonctionnelle

Dans une volonté de démarche agile, nous souhaitons permettre grâce à cette proposition de POC (*Proof of Concept*) à notre commanditaire, la direction du S.H.I.E.L.D. , de se projeter rapidement dans l'utilisation de l'application. Le module de déclaration des incidents a été choisi pour cette maquette car il s'agit de la première étape du processus métier principal de l'application : la gestion des interventions des équipes de Super-Héros. En combinant l'itération de cycle de livraison rapide à nos études techniques et fonctionnelles, nous pourrons être plus réactifs dans la prise en compte des retours clients tout en garantissant la stabilité et l'évolutivité du C.R.M.

## 4 La modélisation UML

### 4.1 Analyse de la mission

Dans cette partie nous avons pour objectif l'analyse UML (Langage de modélisation unifiée) du projet. Pour débuter, nous ferons une analyse comparative des différents logiciels de modélisation UML disponibles sur le marché. Pour chaque module de ce projet nous allons réaliser des diagrammes permettant de visualiser toute l'infrastructure ainsi que le mode de fonctionnement que devra respecter notre application.

Nous allons d'abord nous concentrer sur les diagrammes de cas d'utilisation qui nous permettront de représenter de façon graphique ce que le système doit faire, ensuite nous effectuerons les diagrammes de flux ou d'activité qui eux représentent de façon schématique un algorithme de résolution de tâche puis enfin nous conclurons avec les diagrammes de classe qui nous montrerons la modélisation conceptuelle générale de la structure de l'application et sa modélisation détaillée.

## 4.2 Outils utilisés

Afin de choisir la solution la plus adaptée, nous avons choisi de comparer plusieurs outils de modélisation UML/graphique. Ce dernier nous permettra de réaliser les diagrammes nécessaires à l'analyse de notre besoin.

Les outils comparés sont :

- StarUML : Logiciel de modélisation UML qui a été « cédé comme open source » par son éditeur, à la fin de son exploitation commerciale (qui continue), sous une licence modifiée de GPL (licence publique générale GNU). Aujourd'hui la version StarUML V3 n'existe qu'en licence propriétaire. StarUML gère la plupart des diagrammes spécifiés dans la norme UML 2.0. Il est écrit en Delphi et dépend de composants Delphi propriétaires (non open-source). Il permet également de réaliser de la rétro-ingénierie sur un code afin d'en extraire les diagrammes UML correspondants.
- Diagrams.net : diagrams.net (anciennement draw.io) est une technologie open source pour la création de diagrammes d'applications. Il est compatible avec Google Drive et existe en version web et desktop.
- PlantText : Outil en ligne gratuit permettant de générer rapidement des images à partir de texte. Se basant sur le langage PlantUML il est principalement utilisé pour réaliser de la modélisation UML. Il permet la création de diagrammes sans utilisation de la souris.
- Enterprise Architect : logiciel de modélisation et de conception UML, édité par la société australienne Sparx Systems. Couvrant, par ses fonctionnalités, l'ensemble des étapes du cycle de conception d'application, il est l'un des logiciels de conception et de modélisation les plus reconnus. En 2018, *Sparx Systems* publie la version 14 revendiquant 740 000 licences. Il permet également de réaliser de la rétro-ingénierie sur un code afin d'en extraire les diagrammes UML correspondants tout comme StartUML.
- Visual Paradigm : Outil UML d'ingénierie logiciel permettant la génération de code à partir de diagrammes UML dans divers langages de programmation. Tout comme Enterprise Architect et StarUML, il permet également de réaliser de la rétro-ingénierie.

Langage	StarUML	Diagrams.net	PlantText	Enterprise Architect	Visual Paradigm
Plateforme	★ Windows, Mac et Linux	★ Windows, Mac, Linux + Web	☆ Web Uniquement	★ Windows, Mac et Linux	★ Windows, Mac et Linux
Type	Logiciel de modélisation UML	Logiciel de modélisation graphique de diagrammes	Logiciel de modélisation de diagrammes par texte	Logiciel de modélisation UML	Logiciel de modélisation UML
Langages de programmation / formats pris en charge	★ C/C++/Java	★ Aucuns	☆ PlantUML	★ ActionScript/ C /C# /C++/ JAVA / PHP /etc.	★ Java/ C# /PHP /etc.
Liberté de mise en forme	☆ Peu importante	★ Très importante	☆ Très réduite	☆ Moyenneme nt importante	★ Assez importante
Version Free	★ Oui	★ Oui	★ Oui	☆ Essai 30 Jours	★ Oui

#### Conclusion :

Nous avons choisi d'utiliser Diagrams.net afin de réaliser nos diagrammes UML en adéquation avec les différents besoins que nous avions listés, présentés ci-après :

- Présentation graphique facilement compréhensible.
- Multiplateforme (Diagrams.net ayant également une version web).
- Pas de besoin de génération de code, notre projet étant consacré au langage Java.
- Simplicité d'utilisation (pas de langage particulier à apprendre).
- Gratuit.
- Facile à prendre en main aux vues des délais et du temps disponible à la réalisation du projet.

Notre groupe ayant déjà utilisé l'outil, notre recul sur celui-ci nous a permis de déterminer que Diagrams.net était tout à fait adapté à notre besoin. Bien que des logiciels comme StarUML apportent beaucoup de fonctionnalités, ces dernières étaient, dans ce cas précis, superflus.

Sources utilisées pour le benchmark :

<https://www.ionos.fr/digitalguide/sites-internet/developpement-web/outils-uml>  
<https://www.wikipedia.com>  
<https://staruml.io>

<https://www.diagrams.net>  
<https://www.planttext.com>  
<https://progsoft.net/fr/software/planttext>  
<http://plantuml.com/fr/guide>  
<https://sparxsystems.com/products/ea/index.html>  
<https://www.visual-paradigm.com>

## 4.3 Les différents diagrammes

### 4.3.1 Diagramme de cas d'utilisation macro

Ce diagramme offre une vision générale des différents cas d'utilisation via notre application. En effet, nous pouvons y voir, pour chaque profil d'utilisateurs, les actions qu'il est en capacité de réaliser.

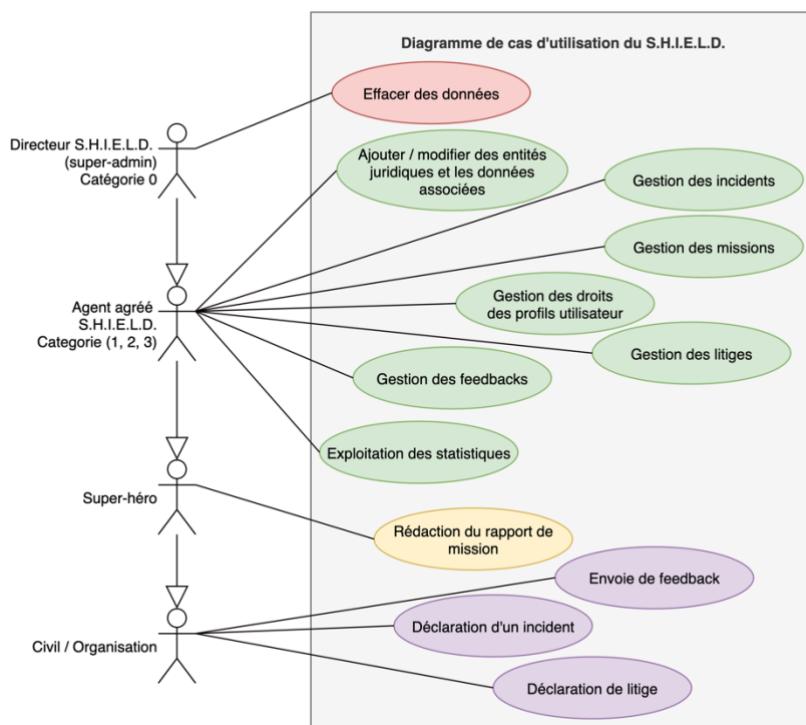


Figure 6 : Diagramme de cas d'utilisation macro

#### 4.3.2 Diagramme de cas d'utilisation micro

Ce diagramme se concentre sur l'utilisation de l'application par les employés du S.H.I.E.L.D. Il illustre le système d'accréditations ascendantes mis en place pour la gestion des droits : chaque niveau hiérarchique hérite des droits d'utilisation des catégories sous-jacentes.

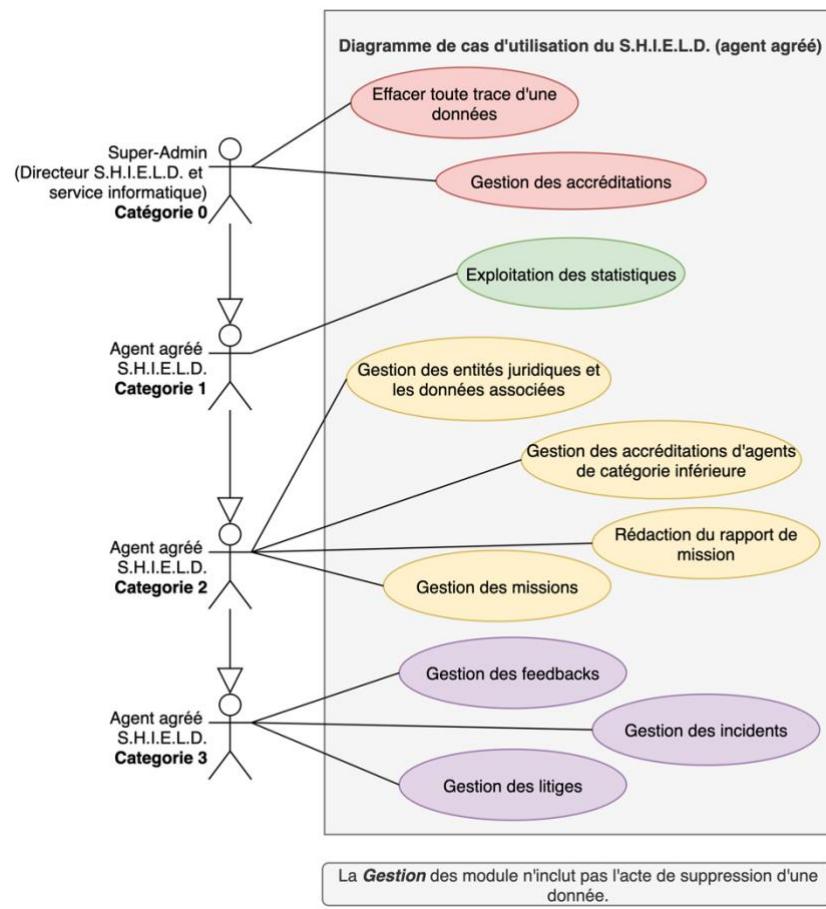


Figure 7 : Diagramme de cas d'utilisation micro

#### 4.3.3 Diagramme de flux du cycle de vie d'une donnée

Afin de continuer l'analyse des besoins de notre application, nous avons réalisé une étape préliminaire dans la conception du système, des diagrammes de flux.

Ces diagrammes consistent en la représentation graphique du traitement des données.

Nous suivons, dans l'exemple suivant, le cycle de vie d'une déclaration d'incident ainsi que celui d'une mission, identifiant ainsi les possibilités d'évolutions et de traitement s'offrant à chaque étape de sa vie. Cela nous permet de mettre en évidence les traitements métiers que nous devons mettre en place.

Les colonnes verticales représentent les « acteurs » réalisant des « actions » sur la donnée, celles horizontales représentent les étapes d'évolution de la donnée.

La bulle verte représentant l'entrée du diagramme et les bulles rouges les sorties possibles.

Les losanges bleus représentant les étapes où la donnée peut suivre un traitement alternatif. Elle ne peut être traitée que sous forme de « oui » ou « non ». Ces étapes sont donc conditionnées dans notre cas par l'intervention d'un type d'acteur.

Les rectangles jaunes représentent également une action nécessitant l'intervention d'un type d'acteur prédéfini, cette fois-ci, sans possibilité de traitement alternatif. Les cartes orange représentant la génération de documents (type PDF etc.).

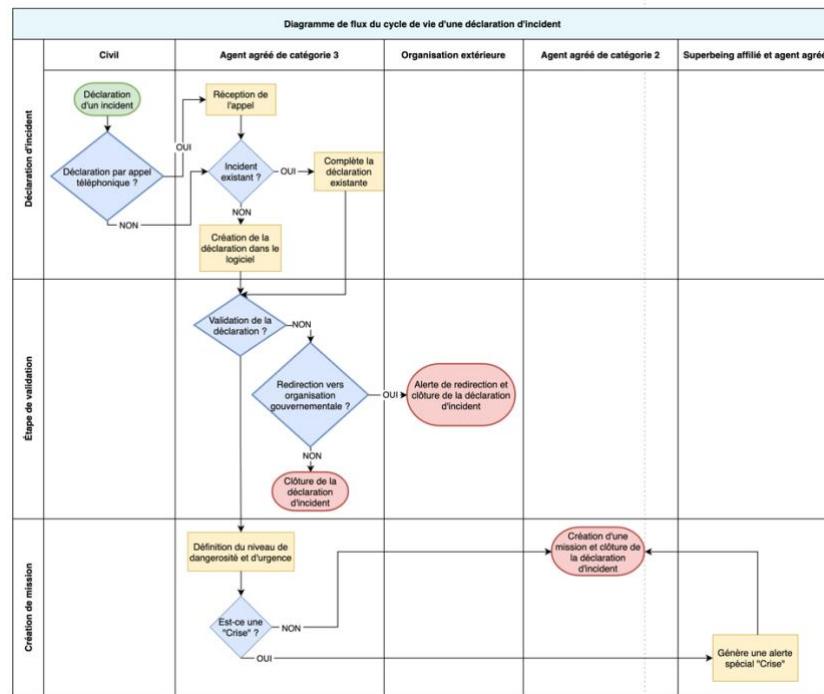


Figure 8 : Diagramme de flux 1

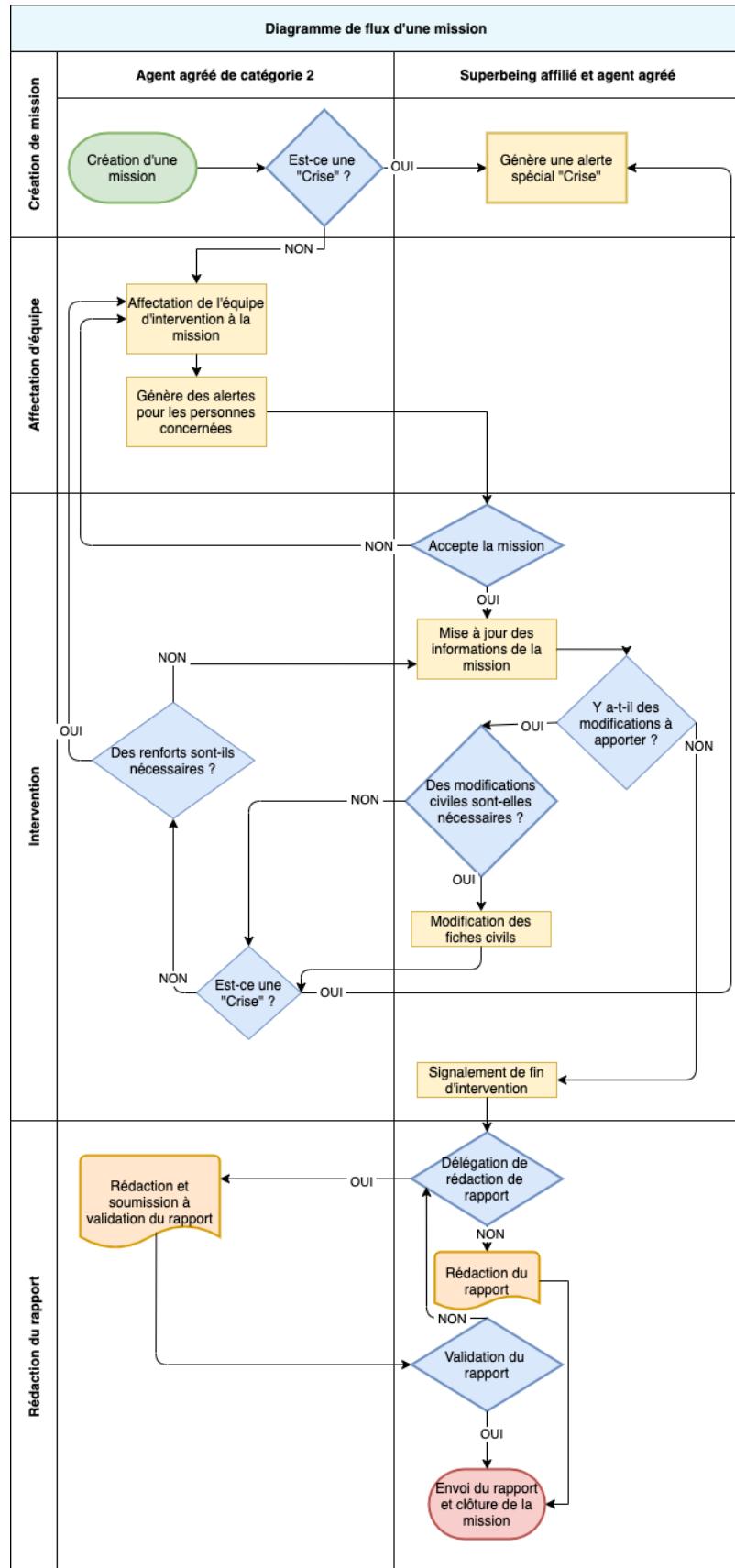


Figure 9 : Diagramme de flux 2

#### 4.3.4 Diagramme de classe

Toujours dans notre objectif d'analyse, nous réalisons maintenant un diagramme de classe. Ce schéma est utilisé pour représenter les entités d'un projet ainsi que leur relation entre eux. Nous avons volontairement omis les « Getter() » et « Setter() », ces méthodes étant des méthodes par défaut devant être présentes au sein de chaque classe. Ce qui retient notre attention est la partie Event et Being, piliers de l'application. En effet nous avons décidé de regrouper la partie de « Event » en un bloc découlant d'une classe principale. La classe « Event » est donc la classe parente ; en découle les classes d'héritage « Incident », « Litige » et « Mission ». Ces classes sont représentées dans la base de données en trois tables différentes à la différence de la partie Being, qui elle regroupe donc les classes « Civil » et « Organisation » mais qui sont représentées dans la base de données en une seule table.

D'autres classes que l'on peut qualifier d'importantes sont les classes avec « connexion multiple » comme « Comment », « Location », « Feedback » et « Media ». Ces dernières sont reliées à quasiment toutes les entités de notre application.

Les classes présentées ci-dessous représentent les entités de notre application. Les classes qui ne représentent pas une réalité métier mais un besoin technique, telles que les Services, Repository ou Controller ne sont pas présentées dans ce diagramme.

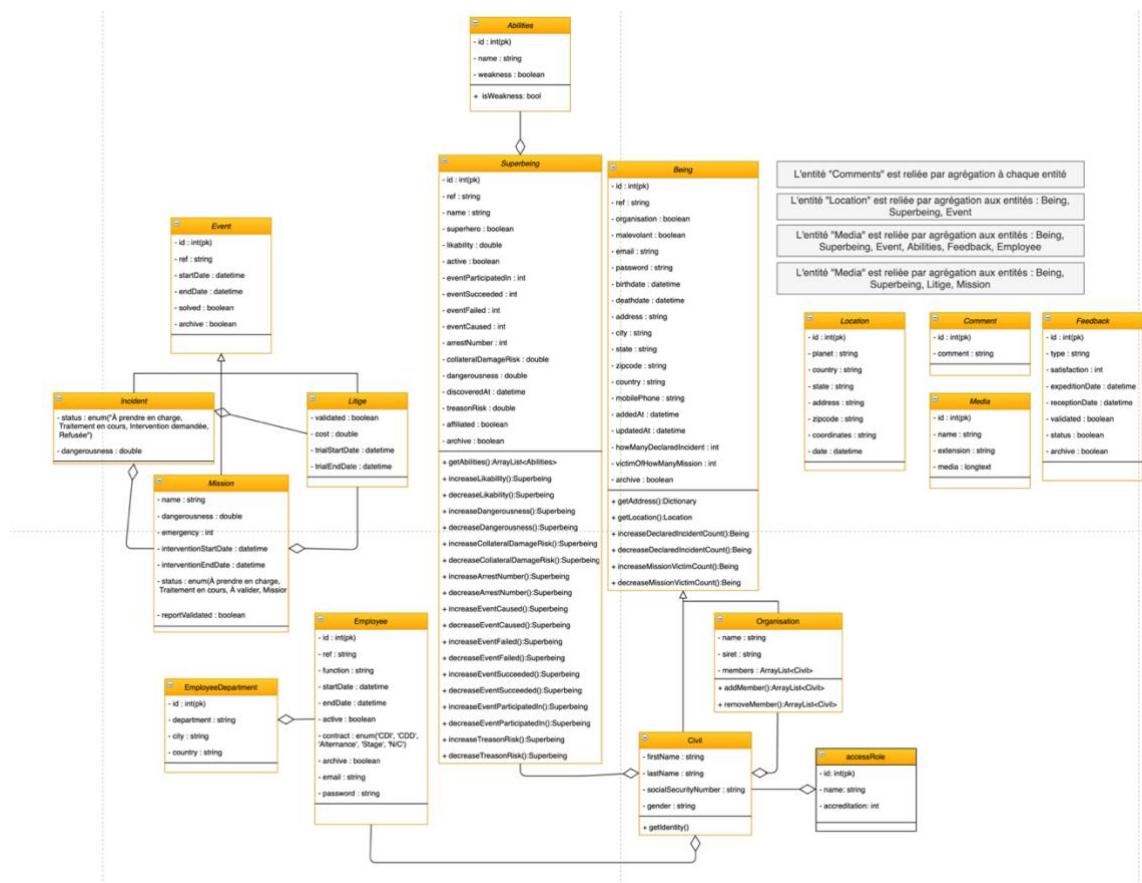


Figure 10 : Diagramme de classe

## 5 La conception

### 5.1 Analyse de la mission

Dans un premier temps, nous allons présenter la conception du design de l'application au travers des outils que nous avons utilisés, de la présentation de notre charte graphique ainsi que la réalisation de la maquette.

Par la suite, et grâce à l'analyse du besoin et à la modélisation UML réalisées en amont, nous avons maintenant la connaissance métier suffisante pour construire notre base de données.

Nous verrons tout d'abord les outils que nous avons choisi et les arguments qui ont motivés nos choix.

Nous présenterons ensuite, à travers les schémas MCD et MLD, notre réflexion sur l'architecture de la base et les décisions prises pour organiser les données de notre CRM.

Après cette étape de conception, nous nous pencherons sur la réalisation concrète de la base de données physiques.

Enfin, une partie axée sur la sécurisation viendra clore ce chapitre consacré à la base de données.

Pour terminer, nous présenterons l'architecture logiciel que nous avons choisi de mettre en place.

### 5.2 Définition du design

#### 5.2.1 Outils utilisés

Pour la conception et la création de la maquette nous avons sélectionné l'outil Adobe XD pour les raisons suivante :

- Il propose une collaboration en direct sur un même projet
- Il est possible de partager des maquettes sous deux modes différents :
  - Développeur
  - Utilisateur / Spectateur (ce qui facilite les démonstrations client)
- La gestion des animations est bien développée et constitue un avantage certain quant à la capacité du spectateur à pouvoir se projeter dans la maquette.

#### 5.2.2 Définition de la charte graphique

La mission de l'entreprise :

Le shield dispose d'une force d'intervention composée d'agents et de super-héros permettant de résoudre les incidents.

Les missions du shield sont :

- De recevoir et traiter les signalements d'incidents
- Effectuer des missions d'intervention afin de résoudre les incidents
- D'identifier et localiser les potentiels ennemis
- Établir une surveillance généralisée de la population à l'aide de critère spécifiques et d'outils statistiques

Le logo de l'entreprise :

Le logo sera utilisé dans deux tailles différentes.

Une grande taille pour la page d'accueil dont les dimensions sont :

- 182\*182 px

Ainsi qu'une petite taille pour toutes les autres pages dont les dimensions sont :

- 60\*60 px

Le logo devra être espacé du reste du contenu afin qu'il soit visible.

Il sera composé de 4 couleurs :

A3A2A3 Quicksilver	FFE104 Golden Yellow	0E1D61 Royal Blue Dark	FFFFFF White	000000 Black
-----------------------	-------------------------	---------------------------	-----------------	-----------------



Les couleurs de l'entreprise :

Voici les couleurs que nos développeurs utiliseront sur le site. Nous avons choisi une palette de couleur dans les bleus avec un contraste rouge pour rappeler les couleurs du drapeau américain.

0F2043 Oxford Blue	FFFFFF White	F2F2F2 Cultured	9E171E Ruby Red	2D61C8 Flickr Blue
-----------------------	-----------------	--------------------	--------------------	-----------------------

La typographie de l'entreprise :

La typographie de l'application vise la "clarté" et à améliorer l'expérience utilisateur.

Le côté « comics » de Marvel est un clin d'œil aux origines de la licence.

Nous utiliserons les polices :

- Marvel pour les tags de titre
- Noto Sans JP pour le contenu de l'application
- Sans-serif et arial comme police de secours

La voix de l'entreprise :

Le sentiment que nous souhaitons communiquer porte sur la sécurité que nous apportons aux citoyens.

### 5.2.3 Réalisation de la maquette

Nous avons privilégié une atmosphère professionnelle adaptée à un service de secours tout en se permettant un clin d'œil autour de l'univers Marvel. Nous nous sommes donc inspirés de l'univers des Comics pour la police des boutons et de la barre de navigation. Concernant le côté administration de l'application, il est important que l'employé du S.H.I.E.L.D. ne soit pas distrait par un design futile, nous avons donc opté pour le côté clair et simple. Vous pouvez retrouver la maquette disponible en téléchargement au lien suivant :

<https://github.com/Kura33/ProjectJavaCesi/master/annexes/maquette>

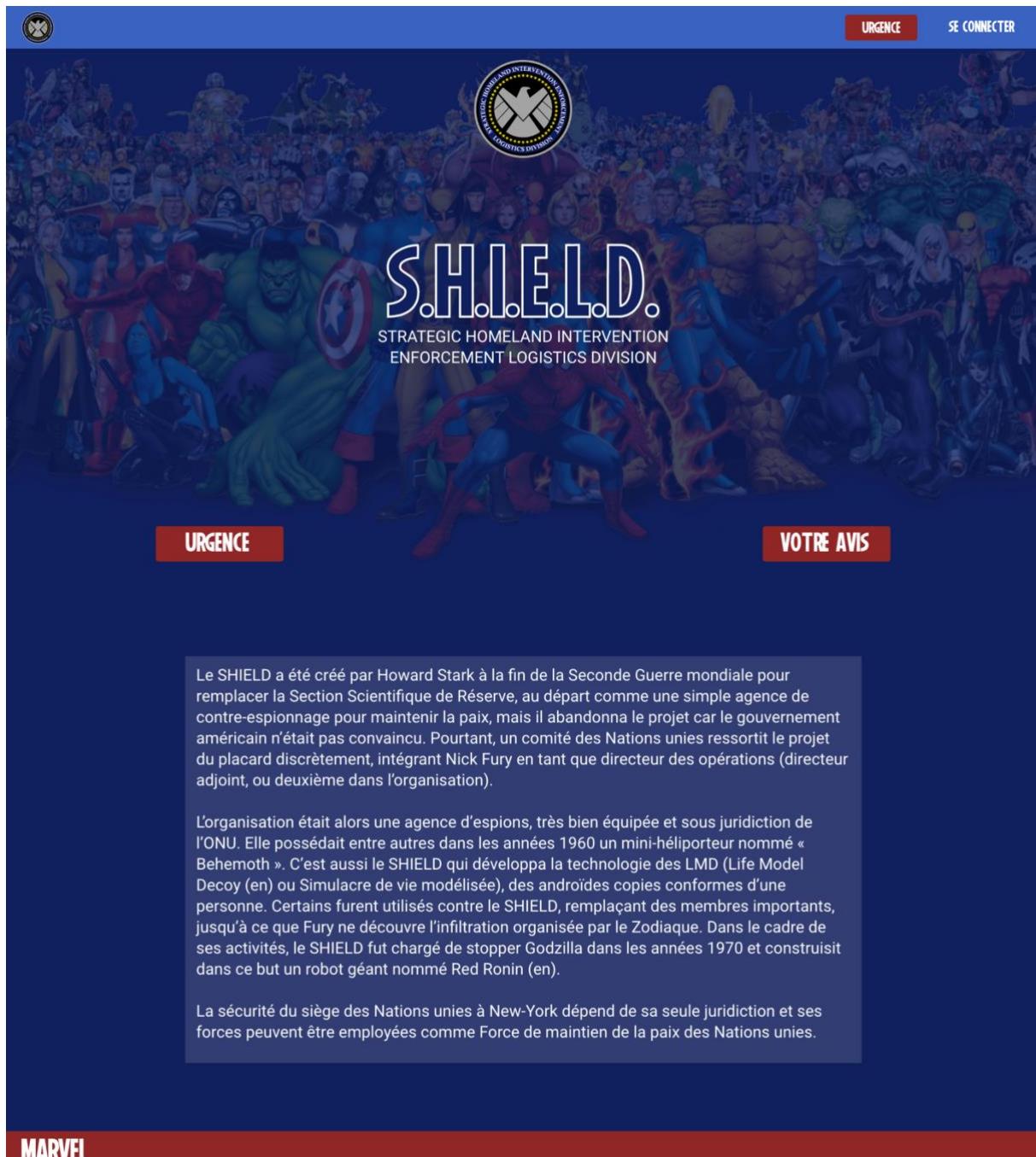


Figure 11 : Aperçu n°1 de la maquette



INCIDENTS

LITIGES

MISSIONS

INCIDENTS

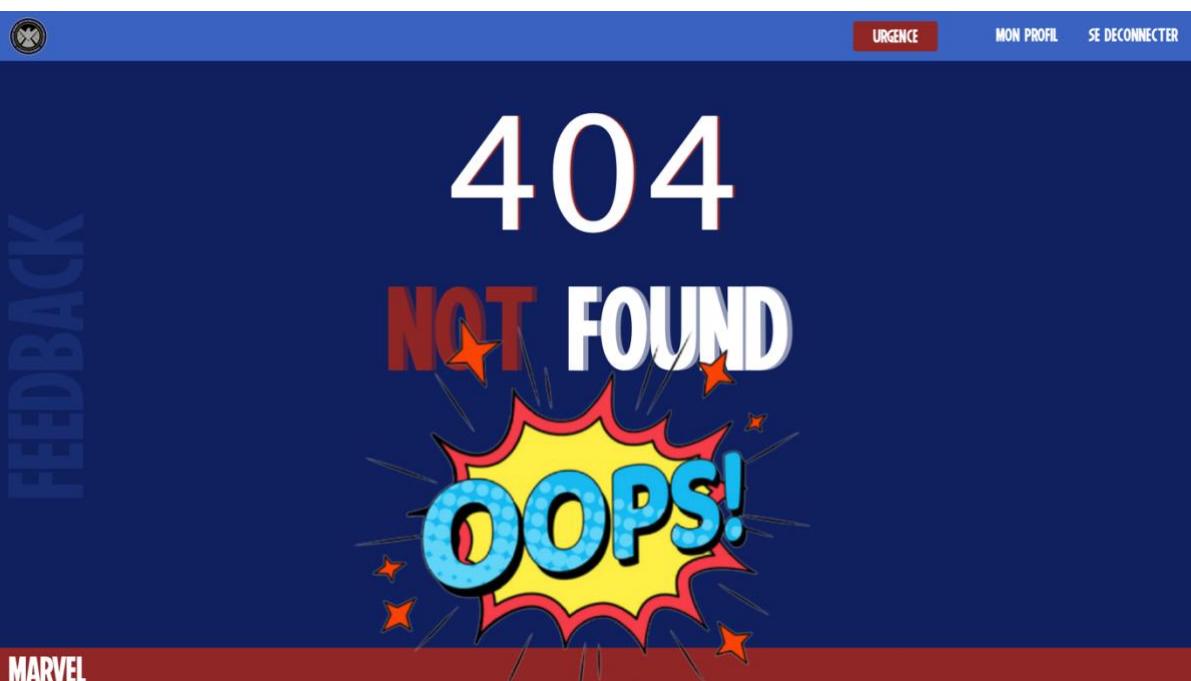
MARVEL

Aperçu n°2 de la maquette

Référence	Timeframe de l'incident	Statut	Dangérosité	Type d'incident	Résolu	Actions
20210121001	21/01/2021 - 27/02/2021	Traitemenent en cours	50%	Incendie	●	ⓘ
20210121001	21/01/2021 -	À prendre en charge	99%	Invasion	●	ⓘ
20210121001	21/01/2021 - 27/02/2021	Intervention demandée	50%	Cambriolage	●	ⓘ
20210121001	21/01/2021 - 27/02/2021	Traitemenent en cours	50%	Incendie	●	ⓘ
20210121001	21/01/2021 -	À prendre en charge	99%	Invasion	●	ⓘ
20210121001	21/01/2021 - 27/02/2021	Intervention demandée	50%	Cambriolage	●	ⓘ
20210121001	21/01/2021 - 27/02/2021	Traitemenent en cours	50%	Incendie	●	ⓘ
20210121001	21/01/2021 -	À prendre en charge	99%	Invasion	●	ⓘ
20210121001	21/01/2021 - 27/02/2021	Intervention demandée	50%	Cambriolage	●	ⓘ
20210121001	21/01/2021 - 27/02/2021	Traitemenent en cours	50%	Incendie	●	ⓘ
20210121001	21/01/2021 -	À prendre en charge	99%	Invasion	●	ⓘ
20210121001	21/01/2021 - 27/02/2021	Intervention demandée	50%	Cambriolage	●	ⓘ

FEEDBACK

MARVEL



## 5.3 Conception et mise en place de la base de données

### 5.3.1 Outils utilisés

Pour la conception et la mise en place de la base de données nous avons choisi des outils spécifiques en adéquation avec nos besoins :

- Pour la création du MCD nous avons utilisé l'outil [Draw io](#). Cet outil offre une grande liberté d'utilisation quant aux options de mises en forme. Par ailleurs, aucune limitation ne viendra nous impacter lors de la création d'entités. La possibilité de travailler en simultané avec ses coéquipiers est également un avantage important.
- Pour la création du MLD nous avons utilisé l'outil [MySQL Workbench](#). Les problèmes connus de l'application sont :
  - Une prise en main complexe
  - Son importante consommation de ressources
  - Le logiciel plante souvent.  
Nous avons donc choisi l'outil pour son côté pratique et sa génération automatique de script malgré ses inconvénients car notre projet ne demandera pas une exploitation trop importante du logiciel.
- Pour la création du MPD nous avons utilisé l'outil [phpMyAdmin](#) malgré une ergonomie austère et ce pour quatre raisons principales :
  - Sa facilité d'installation, notamment sur un serveur
  - Sa symbiose avec MySQL
  - Sa popularité permet l'accès aux nombreuses ressources le concernant
  - C'est un outil que nous maîtrisons et utilisons dans le domaine professionnel.

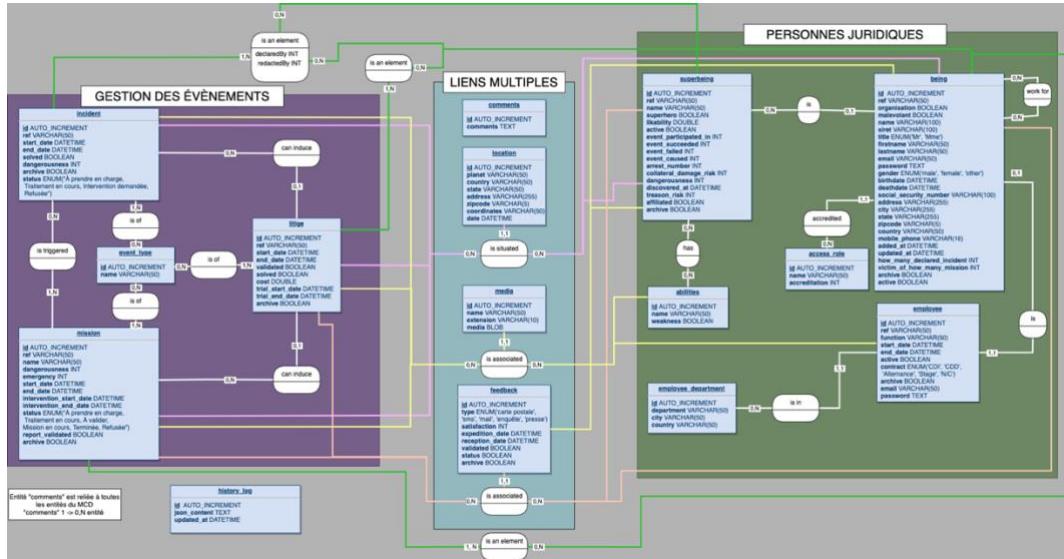
### 5.3.2 MCD / MLD

Maintenant que nous avons analysé l'ensemble de nos besoins, nous pouvons désormais concevoir notre base de données. Pour commencer, nous avons créé notre Modèle Conceptuel de Données présenté ci-après.

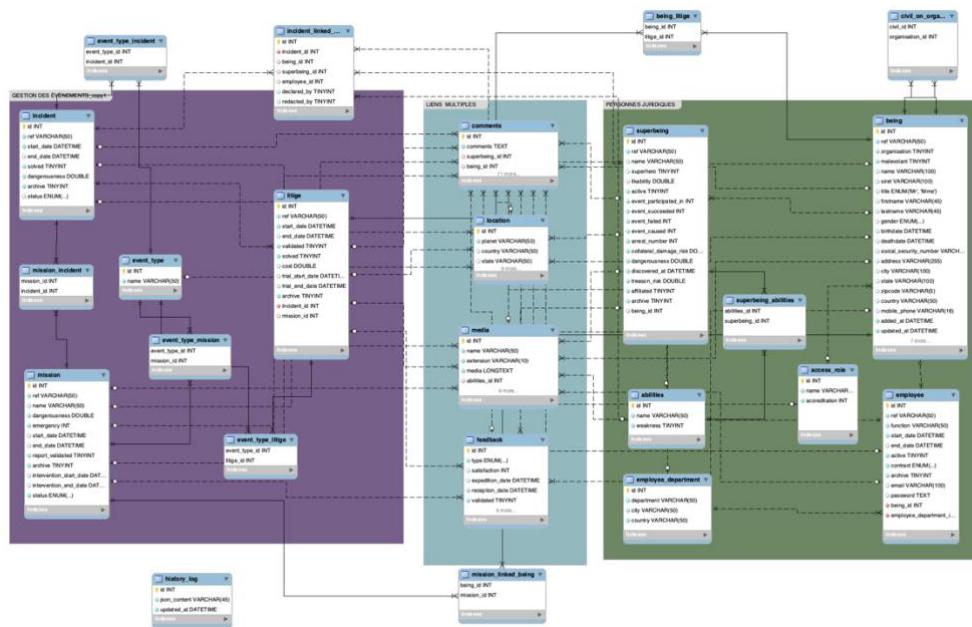
Notons les particularités que nous avons mis en place afin de répondre aux besoins définis précédemment :

- La table « being » regroupe les différentes entités Civils et Organisation. Leur différentiation s'effectuera par un booléen « organisation » où TRUE signifiera que l'entité est une organisation.
- Une association « work for » bouclant sur la table « being » permet d'associer des civils avec une organisation.
- La gestion des droits s'effectuera grâce à une relation avec la table « access\_role » définissant des niveaux d'accréditation (0, 1, 2, 3, 4).
- La table « employee » stockera le login (email) et le mot de passe de connexion de chaque employé du S.H.I.E.L.D., indiquant également leur fonction et le département où ils sont affiliés.
- La table « superbeing » réunit les Super-Héros et les Super-Vilains. Ils se différencient avec un booléen « superhero » où TRUE signifiera que l'entité est un Super-Héros. De plus, la colonne « affiliated » permet de déterminer si le « superbeing » est affilié au S.H.I.E.L.D.. Un champ « likability » permet de suivre sa popularité.
- Il en va de même pour la table « ability » qui différenciera les faiblesses des pouvoirs/techniques où TRUE signifie que l'habileté est une faiblesse.

- La table « event\_type » permet de catégoriser les différents types d'évènement des incidents, des missions et des litiges (Cambriolages, Destructions de biens, etc.).
- Les tables « media » et « location » ont de nombreuses relations permettant la traçabilité et la jonction de documents (images, vidéos, PDF, etc.).
- La table « comment » est reliée à toutes les entités, permettant ainsi aux utilisateurs d'apporter des précisions sur chaque élément produit.
- La table « history\_log » permettra d'enregistrer toutes les modifications apportées aux données de la base de données.



Nous vous présentons maintenant le Modèle Logique des Données :



### 5.3.3 *Création & sécurisation*

Afin d'héberger notre base de données, nous allons utiliser le logiciel XAMPP (MAMP pour Apple). Ces deux logiciels gratuits nous offrent un service facile d'utilisation que nous avons déjà étudié et expérimenté.

Nous avons donc hébergé notre Base de Données sur PHPMyAdmin : Outil gratuit et pratique que nous utilisons régulièrement et que nous avons présenté par le passé dans nos précédents projets respectifs. Il correspond parfaitement à nos besoins simples en la matière : un outil gratuit et facile d'utilisation.

Suite à la réalisation de notre MLD, nous avons généré un script SQL avec les instructions permettant d'injecter des données. Le script étant disponible sur notre [GitHub](#).

Afin de sécuriser la Base de Données, nous devrons changer le mot de passe du compte root par un mot de passe sécurisé, nous pourrions également changer le nom du compte root pour limiter les risques. Par la suite nous devrons créer un compte administrateur avec un nouveau mot de passe sécurisée. Nous devrons effectuer la connexion à la Base de Données avec ce compte administrateur.

## 5.4 Architecture logicielle

### 5.4.1 *Présentation de l'interface de programmation d'application*

Une API ou « *Application programming interface* » représente un ensemble de définitions et de protocoles qui facilite la création et l'intégration de logiciels d'applications. Pour faire simple, c'est un messager qui emmène une requête vers un système, lui explique en quoi consiste cette requête, et nous rapporte une réponse. C'est grâce à cette logique que notre application va pouvoir interagir avec nos utilisateurs. En effet c'est avec celle-ci que nous créerons les différents **endpoints** (Un **Endpoint** est ce qu'on appelle une extrémité d'un canal de communication. Cela représente donc le point de contact que rejoint le message avec sa requête).

Lors de la définition de notre architecture, nous avons dressé la liste des APIs que nous devions développer et leurs spécifications.

Avant la mise en place de Swagger, nous avons commencé notre réflexion dans un document Word. Au fur et à mesure de notre avancée dans la connaissance du Framework Spring Boot et suite à quelques réajustements métiers, nous avons fait évoluer nos spécifications dont la documentation actuelle se trouve aujourd'hui dans Swagger.

*Voici un extrait de l'ébauche des spécifications api, où l'on peut notamment voir une première réflexion en prévision des futures fonctionnalités de tri et de pagination de la liste d'incident :*

## /shield/incident

### GET

#### Description

Return incidents list.

#### Paramètres

NAME	IN (QUERY, PATH, BODY)	DESCRIPTION	REQUIRED	TYPE
nbResultsByPage	Query	Number of result to display	True	Integer
sorting	Query	Sorting criterion &sort=[-start_date,-dangerousness ]	False	List
filter	Query	Filter criterion (KHS Brackets) &sort=[validated[eq]=true,solved[eq]=true ]	False	List
currentPage	Query	Page to display	True	Integer

### BODY

#### Exemples réponses

##### Valide

Code de retour : « 200 »

Description :

```
[  
  {  
    "id": 1,  
    "ref": "20210121001",  
    "startDate": "2021-01-21",  
    "endDate": "2021-02-27",  
    "solved": "False",  
    "dangerousness": 2,  
    "event_type": "Cambriolage",  
    "status": "Pris en charge",  
  },  
  {  
    "id": 2,  
    "ref": "20210121002",  
    "startDate": "2021-01-21",  
    "endDate": "2021-01-2",  
    "solved": False,  
    "dangerousness": 0,  
    "event_type": "Coup d'état",  
    "status": "Refusée",  
  },  
]
```

##### Invalide

Code de retour (exemple): 400

Description :

```
{  
  "code": 400,  
  "message": "Paramètre Id manquant."  
}
```

Figure 16 : Specs API préparatoires

Voici la documentation Swagger de l'API actuellement livrée :

The screenshot shows the Swagger UI for a GET request to the endpoint `/shield/incident/findAllIncidents`. The 'Parameters' section indicates 'No parameters'. The 'Responses' section shows a 200 OK response with a sample JSON payload:

```
[
  {
    "dangerousness": 0,
    "endate": "2021-03-27T17:18:50.217Z",
    "eventType": "string",
    "id": 0,
    "ref": "string",
    "solved": true,
    "startDate": "2021-03-27T17:18:50.217Z",
    "status": "string"
  }
]
```

Figure 17 : Specs API actuelles de Swagger

### Sources utilisées pour la réalisation des spécifications API :

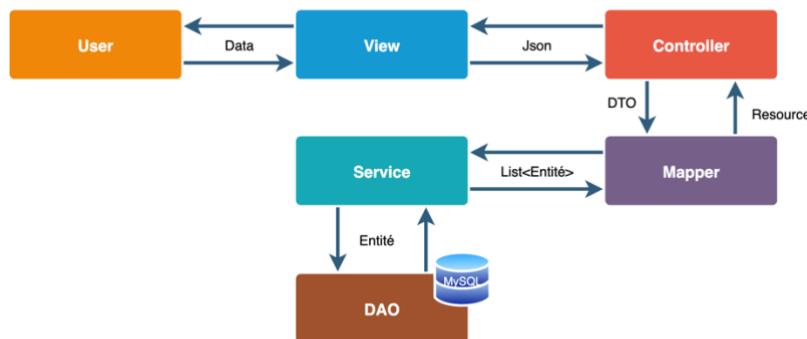
<https://swagger.io/bloq/api-design/api-design-best-practices/>

<https://medium.com/sorting-filtering-retrieving-specific-fields-in-your-api>

<https://christiangiacomi.com/posts/rest-design-principles/>

#### 5.4.2 Définition de l'architecture

Concernant notre pattern design, nous nous sommes appuyés sur un modèle MVC (Model View Controller). Cependant, dans notre projet, le Model « classique » est séparé en plusieurs classes qui permettent une gestion des flux plus précise et facilitera la modification d'une partie du code en limitant l'incidence du couplage entre les classes. Voir ci-contre le schéma de notre design pattern :



- **ENTITE**

L'entité est une classe représentant un concept métier clef de l'application. Ses attributs sont directement mappés avec le modèle de données en base.

*Exemple : dans le workflow de l'application, une demande d'incident peut entraîner la création d'une mission. Mission et Incident sont donc deux classes métiers.*

Raisonner l'architecture en objets métier permet *by design* d'intégrer les besoins des utilisateurs au cœur de la logique de développement et ainsi faciliter l'intégration de nouvelles règles métier par la suite.

- **DTO**

Le DTO (Data Transfer Object) est un design pattern qui permet, comme son nom l'indique, de stocker et transférer des données. Il s'agit d'un intermédiaire entre la couche de visualisation et les classes métiers. Ses attributs définissent la structure des données qui seront échangées.

L'utilisation de DTO complexifie le code et la nécessité de traverser une couche supplémentaire à chaque interaction client/serveur. Cependant, ces contraintes sont largement compensées par les avantages obtenus lors du développement d'une application aussi vaste qu'un CRM :

- **Meilleure appréhension des données transitées** : les données transmises à la couche d'affichage sont facilement identifiables par la consultation du DTO.
- **Amélioration générale de la sécurité et de l'évolutivité de la solution par encapsulation des données.** L'impact d'une modification de la base de données sur la couche de visualisation est minimisé par l'utilisation des DTO.

- **RESOURCE**

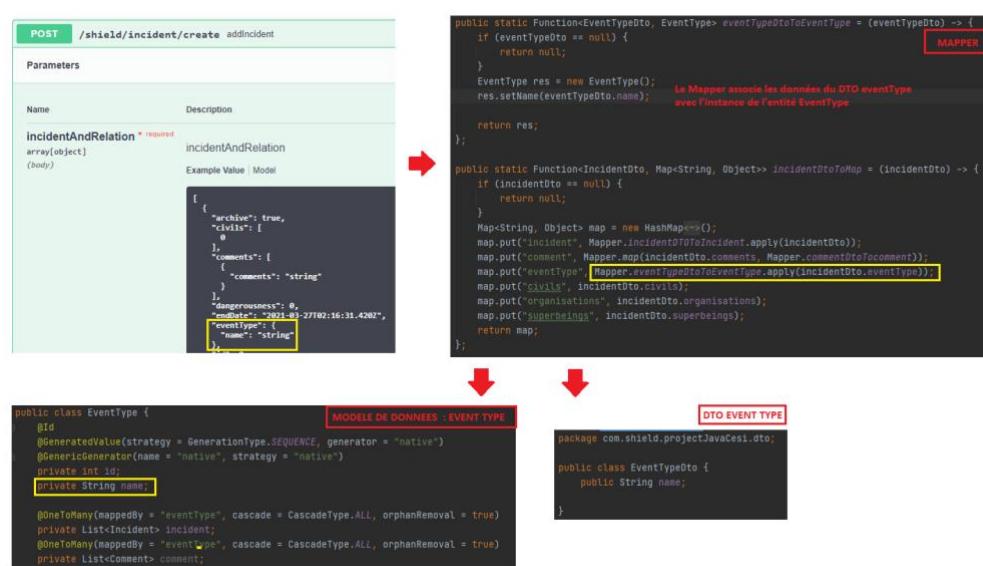
Nous avons systématiquement dissocié les objets de transfert entrant (du front vers le serveur) et sortant (du serveur vers le front), même lorsque les données attendues étaient identiques. Cela garantira l'évolutivité de notre système si par la suite, la structure des données à transiter devait évoluer vers des attendus différents en entrée et en sortie.

Les DTO stockent les données en entrée c'est-à-dire les données attendues du front (opérations de création ou d'update). Les Resources stockent les données en sortie soit celles en provenance du serveur (opérations de lecture).

- **MAPPER**

La classe Mapper assure la correspondance entre l'entité et le DTO ou la Ressource en assignant aux attributs de la première, les valeurs des attributs du second.

Le schéma ci-dessous illustre l'articulation entre ces différentes classes :



- **REPOSITORY :**  
C'est une interface dans laquelle nous effectuerons les appels / requêtes à la base de données. Elle hérite des méthodes par défaut de JPA Hibernate.
- **SERVICE :**  
Le service sert de passe-plat entre le repository et le controller. Le service doit fournir une « API » à votre logique métier, étant donc une abstraction de votre repository. Les services doivent être les seuls à avoir accès à ceux-ci, sinon cela viole le [Principe d'inversion de dépendance](#) (SOLID Design) car la couche de gestion est une abstraction de votre couche d'accès aux données. Les principes de conception SOLID ont été promus par Robert C. Martin et font partie des principes de conception les plus connus dans le développement de logiciels orientés objet. SOLID est un acronyme mnémotechnique pour les cinq principes suivants :
  - Principe de responsabilité unique (*Single-Responsibility Principle*)
  - Principe ouvert / fermé (*Open/Closed Principle*)
  - Principe de substitution de Liskov (*Liskov Substitution Principle*)
  - Principe de séparation des interfaces (*Interface Segregation Principle*)
  - Principe d'inversion de dépendance (*Dependency Inversion Principle*)
 Chacun de ces principes peut être autonome et a pour objectif d'améliorer la robustesse et la maintenabilité des applications et des composants logiciels orientés objet. Mais ils s'ajoutent aussi les uns aux autres de sorte que leur application rend la mise en œuvre de chaque principe plus facile et plus efficace.
- **CONTROLLER :**  
Les classes Controller fonctionnent comme une passerelle entre votre entrée et la logique de domaine, décident quoi faire avec la requête et comment renvoyer la réponse. Un Controller contient très peu de logique et est utilisée pour effectuer des appels aux services. Il effectue rarement des appels directs aux repository, sauf s'il existe une raison valable. Le Controller effectuera des vérifications de base sur les données renvoyées par les services afin de renvoyer une réponse au client.
- **VIEWS :**  
Cela correspond à la partie visible de l'application, la partie à laquelle l'utilisateur aura accès. La view se sert des informations que lui passe le controller pour gérer l'affichage des données. De plus c'est de la view que les nouvelles requêtes faites par l'utilisateur démarreront.

Sources :

[https://www.e-bancel.com/architecture\\_metier\\_classes\\_collections.php](https://www.e-bancel.com/architecture_metier_classes_collections.php)  
<https://www.developpez.net/forums/d1943757/java/general-java/persistancedonnees/difference-dao-dto/>  
<https://www.infoq.com/fr/articles/dto-hipster-ou-depasse/>

# 6 Réalisation technique

## 6.1 Analyse de la mission

Nous allons désormais vous présenter notre partie technique.

Dans un premier temps, nous analyserons les outils que nous avons utilisés afin de réaliser notre projet (IDE, Framework, librairie etc...).

Puis, nous présenterons notre stratégie de versionnage suivi des règles de standardisation ainsi que l'infrastructure qui a été nécessaire à la réalisation technique du projet.

Nous présenterons en suite les points clés de notre application.

Pour clôturer la partie technique, nous avons réalisé des tests unitaires ainsi qu'une fiche de Test Qualité.

## 6.2 Outils utilisés

Nous allons maintenant vous présenter les outils que nous avons choisi d'utiliser afin de réaliser notre application.

### 6.2.1 *Les langages*

Tout d'abord le langage que nous avons utilisé en backend :

- Java, sujet de notre projet étudiant, Java est un Langage de programmation orienté objet qui va nous permettre de développer l'API que nous avons réalisé.  
Les langages que nous avons choisis afin de réaliser le front-end :
  - Html5/ CSS3/ JavaScript, la base de tout projet ayant une interface web. Le langage JavaScript étant essentiellement destiné au front-end et la multitude de ses Framework fait de lui un outil tout à fait adéquat a notre projet.  
En ce qui concerne le format de données textuel :
    - JSON (JavaScript Object Notation). Format simple et intuitif à apprendre remplaçant petit à petit le langage XML. Le fait qu'il soit écrit en JavaScript le rend plus facile à utiliser avec le langage JavaScript ou ses Frameworks tel que VueJS.

### 6.2.2 *En ce qui concerne le choix de l'IDE :*

- Eclipse/ IntelliJ IDEA. Nous avons tout d'abord opté pour l'IDE Eclipse, mais suite a quelques soucis techniques avec l'outil git de l'IDE, nous avons optés pour une transition sur IntelliJ IDEA qui nous conviens mieux en ce qui concerne l'UI/UX. Le choix d'un IDE restant assez personnel, il n'y aura aucune problématique dans le cas où certains membres de l'équipes souhaiteraient avoir différents IDE.

### 6.2.3 *Les Framework*

- Springboot est un Framework open source permettant de construire l'infrastructure d'une application Java. D'autres Framework étaient à notre disposition tel que QUARKUS ou Play Framework pour Java, nous avons dû faire un choix arbitraire qui a

été influencé par les conseils de professionnels du métier et la grande disponibilité des ressources sur internet.

- Hibernate est un Framework open source permettant de gérer la persistance des objets en base de données. Son avantage est de masquer la logique relationnelle aux développeurs objets et/ou d'interconnecter "facilement" des objets avec une base de données "business" existante. Springboot intégrant Hibernante et possédant des ressources nombreuses sur internet, notre choix se porta naturellement sur ce Framework.
- VueJS, Framework JavaScript qui va nous permettre de rendre l'affichage du site dynamique. Il est très adapté au développement front Web et de nombreux tutoriels sont disponible au vu de sa communauté active. Notre groupe ayant déjà une certaine appétence pour ce Framework, il fut pour nous logique de l'utiliser.

#### 6.2.4 *Les librairies*

Backend :

- JPA, Java Persistence API est une interface de programmation permettant l'organisation de données relationnelles.
- JWT, signifiant JSON Web Token, est une méthode permettant l'échange sécurisé de jetons(identificateur) entre plusieurs parties.

Front-end :

- Bootstrap est une collection d'outils permettant la mise de place de designs, il contient des outils tels que des boutons, des outils de navigations etc.
- Axios est une bibliothèque JavaScript fonctionnant comme un client http. Elle permet la communication avec les API.
- Moment.js est une librairie permettant de parser, valider, manipuler et formater des données.

#### 6.2.5 *Outils Annexes*

- Apache Tomcat est un conteneur web destiné à Java. Springboot l'intègre
- Apache Maven est un outil de gestion d'automatisation de projet logiciel Java et JEE. Il nous permet d'indiquer les différentes librairies externes que nous souhaitons utiliser dans notre projet. Ainsi lors du « build » de l'application l'utilisateur backend téléchargera automatiquement les différents outils. Bien que Gradle soit plus récent et puissant, il reste plus adapté à des gros projets. Sachant que Gradle s'appuie sur Maven, nous avons donc choisi de débuter sur ce dernier.
- Postman est une plateforme collaborative destiné au test des API. Nous l'utilisons dans notre quotidien professionnel, de plus, il est gratuit et simple d'utilisation.
- Swagger est un outil installable au sein de notre application qui nous permettra de générer la documentation de notre API au format web. De plus, l'outil nous permet d'accéder à un équivalent de Postman en ligne. Il est donc un outil fort apprécié des développeur front-end qui devrons manipuler les endpoints de l'API.

Les choix que nous avons fait en ce qui concerne les différents outils techniques que nous avons utilisé ont été influencés par notre affinité avec ceux-ci. En effet les différentes fonctionnalités apportées par ces outils sont très similaires. Notre

premier besoin étant de disposer d'un serveur dédié à Java, notre choix nous a porté vers Springboot qui contient un server Apache Tomcat. Hibernate étant compris dans ce même outil, le choix de son utilisation était logique.

En ce qui concerne le front-end, notre choix se porta sur une page web accessible à distance grâce à VueJS. Au vu de nos compétences et notre projet étudiant portant sur Java, nous avons souhaité approfondir nos connaissances dans ce langage.

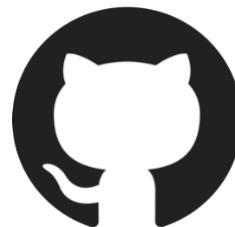
Sources :

<https://blog.dyma.fr/angular-vs-vue-vs-react>  
<https://www.jetbrains.com/fr-fr/idea/>  
<https://www.eclipse.org/ide/>  
<https://vuejs.org/>  
[https://fr.wikipedia.org/wiki/Java\\_Persistence\\_API](https://fr.wikipedia.org/wiki/Java_Persistence_API)  
<https://jwt.io/>  
<https://spring.io/projects/spring-boot>  
<https://quarkus.io/>  
<https://hibernate.org/>  
<https://stackshare.io/hibernate/alternatives>  
<https://www.journaldunet.fr/web-tech/axios>  
<https://momentjs.com/>  
<http://tomcat.apache.org/>  
<https://maven.apache.org/>  
<https://dzone.com/articles/gradle-vs-maven>  
<https://www.postman.com/>  
<https://swagger.io/>

### 6.3 Versionnage

Afin de pouvoir effectuer un travail collaboratif efficace, nous avons choisi de versionner notre production grâce aux outils Git et GitHub. Ils nous permettront de concaténer notre travail ainsi que de pouvoir, en cas d'erreur dans le développement, revenir sur une version précédente plus stable afin de repartir sur une base saine.

Après avoir créé notre projet Java sur notre IDE, nous avons créé le repository de notre projet sur GitHub. Après avoir effectué la liaison entre notre repository local et le repository GitHub, nous avons ajouté les autres membres de l'équipe en tant que collaborateurs, permettant ainsi à tous de pouvoir interagir librement avec le repository.



Notre stratégie de versionnage du développement s'effectuera suivant l'architecture suivante :

Master : branche destinée au produit final.

Dev : branche destinée aux versions stables.

Des branches de développement par thématique telles que « Event », « Being », « Superbeing », « Permission » etc..

Les branches Master et Dev sont protégées par le biais de l'activation des « Pull Request » sur GitHub nécessitant la validation des 2 autres membres de l'équipe. De plus les « Push Force » sont désactivés pour l'administrateur du repository.

Les membres développeront donc la thématique qui leur a été attribuée et lorsque la version développée sera stable, ils pourront faire une demande de « Pull Request » afin d'envoyer leur code vers la branche Dev.

La communication est primordiale si l'on veut avoir une bonne cohérence d'équipe et ne pas se retrouver bloquer dans notre développement.

Vous retrouverez notre projet sur le lien suivant :

<https://github.com/Kura33/ProjectJavaCesi>

#### 6.4 Règles de standardisation

Afin de standardiser notre code, nous avons choisi de suivre le camelCase.

Le nom des différents éléments sera en anglais même si l'affichage front-end sera en français. Cela permettra la portabilité de notre projet à l'internationale. De plus le S.H.I.E.L.D. est une institution américaine.

Les autres règles suivies seront celles du langage JAVA.

Les docstrings (commentaires) permettent de documenter le code :

- Description de la méthode
- Paramètres pris en entrée
- Le type de retour (un objet, une liste d'objet, etc.)
- Les Exceptions si levées.

La dénomination des différents éléments devra être claire quant à l'action qu'ils permettent d'effectuer.

Ajoutons que les attributs déclarés sont tous privés afin d'en interdire la modification. Les seuls attributs publics sont ceux des DTO et Resources, de par leur nature de représentations de données externes.

Le découpage par package de notre application entre également en compte dans notre stratégie de standardisation. Cette partie vous est détaillée dans une autre section de notre présentation.

Toutes ces règles sont des « bonnes pratiques » permettant de limiter les risques d'erreurs dans les appels effectués. Elles permettront de réduire la dette technique ainsi que les coûts et temps de maintenance.

## 6.5 Infrastructure

Le code de l'application back est découpé en packages fonctionnels. Le rôle de la classe déterminera donc son emplacement.

Dans certains packages, un deuxième niveau de découpage selon des regroupements thématiques métiers a été mis en place pour des raisons de lisibilité.

### 6.5.1 Packages fonctionnels

CONFIG	
<i>Toutes les classes de configuration transverses à l'application</i>	
Class	Description
JwtAuthenticationEntryPoint	Jason Web Token Renvoie une erreur en cas de connexion non identifiée
JwtRequestFilter	Jason Web Token Validation du token et mise à jour du contexte utilisateur
JwtTokenUtil	Jason Web Token Méthodes utilitaires de traitement sur le token
TokenResponse	Jason Web Token Classe du token généré
WebSecurityConfig	Cette classe permet de gérer la configuration web, notamment l'activation du système de permission et l'autorisation d'accès aux requêtes http...
SwaggerUiWebMvcConfigurer	Permet de gérer les resources et les controllers Swagger

CONTROLLER
<i>Définit les routes API – Gère les requêtes http entrantes et envoie les réponses au client</i>

DTO
<i>Data transfert object : objet de stockage des données en provenance du client</i>

RESSOURCE
<i>Objet de stockage des données en provenance du serveur consommable par le client</i>

ENTITY
<i>Regroupe les classes métier.</i>

## MAPPER

*Classe unique de mapping entre DTO / Resource et Entity*

## REPOSITORY

*Regroupe les interfaces de requêtage à la base de données héritant de JpaRepository.*

## SERVICE

*Regroupe tous les traitements métiers appliqués aux entités y compris la récupération des permissions à la connexion de l'utilisateur.*

### 6.5.2 Sous-packages thématiques

REGROUPEMENT THEMATIQUES METIERS		
Package	Description	Comprend
BEING	Concerne les entités juridiques d'après leur identification civile et leurs droits d'accès à l'application	<ul style="list-style-type: none"><li>• Classe mère abstraite Being</li><li>• Classes filles Civil et Organisation</li><li>• Classe AccessRole (gestion des droits)</li></ul>
EMPLOYEE	Concerne les employés du S.H.I.E.L.D.	<ul style="list-style-type: none"><li>• Classe EmployeeEmployee</li><li>• Classe EmployeeDepartment (service de rattachement))</li></ul>
EVENT	Rassemble les différents types d'évènements : Incident, Mission et Litige	<ul style="list-style-type: none"><li>• Classe mère abstraite Event</li><li>• Classe EventType (catégorie selon la nature du trouble)</li><li>• Classe fille Incident</li><li>• Classe Mission</li></ul>
MULTIPLE CONNECTION	Regroupement de classes ayant de nombreuses interactions avec les autres packages.	<ul style="list-style-type: none"><li>• Classe Comment</li><li>• Classe media</li><li>• Classe Location</li></ul>
SUPERBEING	Concerne les Super-Héros et / ou les Super-Villains	<ul style="list-style-type: none"><li>• Classe Superbeing</li><li>• Classe Ability</li></ul>

## 6.6 Code

### 6.6.1 Fichier de configuration

Avant toute chose il est nécessaire de mettre en place un fichier de configuration. Ce dernier se nomme « application.properties » et permettra de lier notre code à la base de données. Pour ce faire il est nécessaire de désigner le type de driver que nous utiliserons (Cf. 1 sur le schéma ci-dessous), le nom, le port et les identifiants requis pour accéder à la base de données (Cf. 2 sur le schéma ci-dessous). Nous avons décidé d'afficher chacune de nos requêtes dans la console pour une gestion des erreurs plus simple et vérifier que notre code est correct (Cf. 3 sur le schéma ci-dessous) et nous disons à notre application d'utiliser le framework Hibernate concernant la persistance des données. (Cf. 4 sur le schéma ci-dessous). Ce fichier sera aussi utilisé pour désigner le port sur lequel l'application tournera et la variable temporaire qui servira de secret pour la génération du JWT Token (Cf. 5 & 6 sur le schéma ci-dessous)

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver spring.datasource.url=jdbc:mysql://localhost:3306/shield spring.datasource.username=root spring.datasource.password=mypassword	1. Driver JDBC MySQL pour l'accès à notre BDD 2. Paramètres de connexion à la base
spring.jpa.show-sql=true spring.jpa.hibernate.ddl-auto=update spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect server.port=8080	3. Affichage du SQL en console 4. Configuration Hibernate
jwt.secret=banana	5. Port du serveur 6. Variable temporaire (POC) pour la génération du JW Token

A noter : ce fichier étant propre à chaque développeur, il figure dans le fichier .gitignore qui liste les éléments à ne jamais push sur le dépôt de l'équipe.

### 6.6.2 Présentation du CRUD (incident)

Représentant l'un des piliers de l'application le CRUD est la partie qui communiquera avec le front-end. C'est grâce à ça que nous pourrons effectuer des requêtes vers la base de données. C'est depuis cette classe que nous avons nommé nos endpoints et donc défini le chemin de requête à effectuer pour accéder à certaines données. Au-dessus de chaque méthode de cette classe il est nécessaire de préciser un @tag de type « Mapping ». De plus le @tag « PreAuthorize » permet de limiter l'utilisation de la méthode aux utilisateurs possédant l'accréditation nécessaire.

L'acronyme CRUD veut dire :

- CREATE :

C'est là que nous allons créer nos méthodes pour insérer de nouvelles données dans la base de données. Le @tag requis pour cette méthode est « PostMapping » dans laquelle le seul paramètre est une liste d'incidents. La liste est analysée et traduite pour devenir conforme avec le type de données requises, on passe donc d'un format json à un format objet « Map<String, Object> ». La liste formatée est ensuite sauvegardée au sein de la base de données.

```

/**
 * Sert à créer un ou plusieurs incidents, un ou plusieurs commentaires, un event type
 * et les relations entre incidents/civils/organisations/superbeings
 * @param incidentAndRelation
 * @return une liste d'incidents.
 */
@PreAuthorize("hasAnyAuthority('Agent_0', 'Agent_1', 'Agent_2', 'Agent_3')")
@PostMapping(value="/create")
public List<IncidentResource> addIncident(@RequestBody List<IncidentDto> incidentAndRelation) {
    List<Map<String, Object>> incidentList = Mapper.map(incidentAndRelation, Mapper.incidentDtoToMap);
    List<Incident> saveIncidentList = service.saveIncidentAndRelations(incidentList);
    em.flush();
    for (Incident incident1 : saveIncidentList) {
        em.refresh(incident1);
    }
    return Mapper.map(saveIncidentList, Mapper.incidentToIncidentResource);
}

```

Figure 18 : Méthode Create Incident

- READ :

La partie read regroupe les méthodes pour aller chercher des données depuis la base de données et les renvoyer vers le front. Elles traduisent cette fois-ci les données depuis Objet vers un tableau de String. Elles sont précédées d'un @tag « GetMapping ».

```

/**
 * Sert à récupérer tous les incidents.
 *
 * @return une liste de tous les incidents.
 */
@PreAuthorize("hasAnyAuthority('Agent_0', 'Agent_1', 'Agent_2', 'Agent_3')")
@GetMapping(value="/")
public List<IncidentResource> findAllIncidents() {
    List<Incident> incidents = service.getIncidents();
    return Mapper.map(incidents, Mapper.incidentToIncidentResource);
}

/**
 * Sert à récupérer un incident
 *
 * @param id
 * @return un incident avec sa liste de commentaires, de superbeing, de civils et d'organisation.
 */
@PreAuthorize("hasAnyAuthority('Agent_0', 'Agent_1', 'Agent_2', 'Agent_3')")
@GetMapping(value="/{id}")
public Map<String, Object> findIncidentById(@PathVariable int id) {
    Map<String, Object> map = service.getIncidentById(id);

    map.put("incident", Mapper.incidentToIncidentResource.apply((Incident) map.get("incident")));
    map.put("comments", Mapper.map((List<Comment>) map.get("comments"), Mapper.commentToCommentResource));
    map.put("superbeings", Mapper.map((List<Superbeing>) map.get("superbeings"), Mapper.superbeingToSuperbeingResource));
    map.put("civils", Mapper.map((List<Civil>) map.get("civils"), Mapper.civilToCivilResource));
    map.put("organisations", Mapper.map((List<Organisation>) map.get("organisations"), Mapper.organisationToOrganisationResource));
    return map;
}

```

Figure 19 : Methode Get Incident

- UPDATE :

La partie update sert à modifier les données d'un objet déjà existant dans la base de données et de sauvegarder ces nouvelles valeurs. Ces méthodes sont précédées d'un @tag « PutMapping ». Cette méthode prend un incident en paramètre, on vient juste modifier l'entité existante et on rafraîchit les données pour la mettre à jour grâce à l'interface EntityManager

```

/**
 * Sert à mettre à jour un incident.
 *
 * @param incident
 * @return un incident.
 */
@PreAuthorize("hasAnyAuthority('Agent_0', 'Agent_1', 'Agent_2', 'Agent_3')")
@PutMapping("/update")
public IncidentResource updateIncident(@RequestBody Incident incident) {
    Incident incidents = service.updateIncident(incident);
    em.flush();
    em.refresh(incidents);

    return Mapper.incidentToIncidentResource.apply(incidents);
}

```

Figure 20 : Méthode Update Incident

- DELETE :

La partie delete regroupe les méthodes de suppression de l'API, ce sont les méthodes les plus « dangereuses » puisqu'elles effaceront de la base de données les informations d'un objet ou l'objet en question. Elles sont précédées d'un @tag « *DeleteMapping* ». Ces méthodes prennent en paramètre un id ou pas de paramètre.

```

/**
 * Sert à supprimer une fiche incident.
 *
 * @param id
 * @return "True" si l'opération s'est bien effectuée.
 */
@PreAuthorize("hasAuthority('Agent_0')")
@DeleteMapping("/delete/{id}")
public String deleteIncidentById(@PathVariable int id) {
    return service.deleteIncident(id);
}

/**
 * Sert à supprimer toutes les fiches incidents.
 *
 * @return "True" si l'opération s'est bien effectuée.
 */
@PreAuthorize("hasAuthority('Agent_0')")
@DeleteMapping("/delete-all")
public String deleteIncidents() {
    return service.deleteAllIncident();
}

```

Figure 21 : Méthode Delete Incident

### 6.6.3 Présentation des Tokens

Lors de la configuration des évènements qui doivent se produire lors d'une connexion, nous avons mis en place un système de génération de Token Web en JSON (JWT) afin de gérer la session d'un utilisateur en lui appliquant les droits d'accès qui lui convienne.

```
/*
 * Sert à se connecter
 *
 * @param employeeDto
 * @return un token de connection valable 24h.
 * @throws Exception
 */
@PostMapping(value="/login")
public TokenResponse login(@RequestBody EmployeeDto employeeDto) throws Exception {
    authenticate(employeeDto.getEmail(), employeeDto.getPassword());

    final UserDetails userDetails = userDetailsService
        .loadUserByUsername(employeeDto.getEmail());

    final String token = jwtTokenUtil.generateToken(userDetails);

    return new TokenResponse(token);
}
```

Figure 22 : Méthode du TokenResponse

La génération de Token est déclenchée lors de l'appel à la connexion dans la classe LoginController.

Le principe consistant à envoyer l'identifiant de l'utilisateur (son email), un mot de passe (tel qu'indiqué dans le format du DTO correspondant) et de recevoir un token en échange. Ce qui identifie l'utilisateur pour une validité de 24 heures si et seulement s'il est présent dans la base de données. Ce Token sera utilisable jusqu'à la déconnexion de l'utilisateur. Une fois retourné au front-end, le token sera passé dans le header "Authorization" des requêtes effectuées auprès de l'API.

Une version plus avancée de notre application viendra modifier cet intervalle de temps afin d'augmenter la sécurité. L'utilisateur lui ne se rendrait pas compte de la mécanique qui serait mise en place grâce à un « refresh Token » qui viendrait maintenir le rafraîchissement du Token à condition que l'utilisateur effectue des actions.

```
@Override
public void configure(HttpSecurity http) throws Exception {
    http.
        csrf().disable()
        .authorizeRequests() ExpressionUrlAuthorizationConfigurer<HttpSecurity>.ExpressionInterceptUrlRegistry
            .antMatchers(AUTH_WHITELIST).permitAll()
            .anyRequest().authenticated()
            .and()
        .logout() LogoutConfigurer<HttpSecurity>
            .permitAll()
            .and().exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint) Exception
            .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);
```

Figure 23 : Méthode de la configuration WebSécurisé

Dans notre classe WebSecurityConfig nous appliquons des filtres n'autorisant que les personnes ayant un Token valide à effectuer des requêtes auprès de l'API sur les routes sécurisées.

Ce système mis en place permet de sécuriser l'accès aux données de notre application car il vérifie que l'utilisateur est authentifié.

#### 6.6.4 Présentation des Mapper

Au sein de la classe Mapper nous rassemblons l'ensemble des méthodes nous permettant de transformer un objet ou une liste d'objets en un autre objet ou liste d'objet. Nous utilisons les "Function" Java qui sont des méthodes permettant de recevoir un objet en argument et de le transformer en un autre type d'objet.

Lorsqu'un utilisateur appelle une méthode GET auprès de l'API, le modèle de données qui va être transformé en JSON prendra le format de Resource. Le cheminement de la donnée ira de la récupération dans la Base de Données et de son initialisation dans notre back end en Objet jusqu'à sa modélisation correspondant à la Resource à laquelle la requête fait référence.

Dans le cas suivant, nous présentons notre mapper permettant la transformation des données BDD en données Resource, utilisé dans le cadre d'un GET Incidents.

```
public static Function<Incident, IncidentResource> incidentToIncidentResource = (incident) -> {
    if (incident == null) {
        return null;
    }
    IncidentResource res = new IncidentResource();
    res.id = incident.getId();
    res.ref = incident.getRef();
    res.startDate = incident.getStartDate();
    if (incident.getEndDate() != null) {
        res.endDate = incident.getEndDate();
    }
    res.solved = incident.isSolved();
    res.dangerousness = incident.getDangerousness();
    res.status = incident.getStatus();
    res.eventType = incident.getEventType().getName();
    return res;
};
```

Figure 24 : Incident Ressource

Ci-après la Resource Incident de référence correspondante :

```
public class IncidentResource {
    public Integer id;
    public String ref;
    public Date startDate;
    public Date endDate;
    public Boolean solved;
    public Double dangerousness;
    public String status;
    public String eventType;
}
```

Figure 25 : Class des resources d'un incident

Dans le cadre d'une méthode POST comme la création d'un incident, le cheminement sera inversé. On recevra des données prenant le format défini dans les DTO (Data To Object) qui seront mappés en Objet souhaité au format adéquat, permettant ainsi de les injecter dans la base de données.

Dans l'exemple suivant nous présentons le Mapper de notre méthode POSTCreate Incident

```
public static Function<IncidentDto, Incident> incidentDTOToIncident = (incidentDto -> {
    Incident res = new Incident();
    if (incidentDto == null) {
        return null;
    }
    res.setRef(incidentDto.ref);
    res.setStartDate(incidentDto.startDate);
    res.setEndDate(incidentDto.endDate);
    res.setSolved(incidentDto.solved);
    res.setDangerousness(incidentDto.dangerousness);
    res.setStatus(incidentDto.status);
    res.setArchive(incidentDto.archive);
    return res;
});
```

Figure 26 : Méthode incidentDto

Notre méthode de création d'incident postant, en plus de l'incident, des associations diverses avec des éléments déjà existants. Nous devons pouvoir transformer le modèle reçu au format DTO en objet correspondant. Pour ce faire nous utilisons une "Function" Java que nous avons créé afin de gérer cette problématique. Elle fait elle-même appel à d'autres Mapper afin d'envoyer les données au bon format dans la base de données.

```
public static Function<IncidentDto, Map<String, Object>> incidentDtoToMap = (incidentDto) -> {
    if (incidentDto == null) {
        return null;
    }
    Map<String, Object> map = new HashMap<~>();
    map.put("incident", Mapper.incidentDTOToIncident.apply(incidentDto));
    map.put("comment", Mapper.map(incidentDto.comments, Mapper.commentDtoToComment));
    map.put("eventType", Mapper.eventTypeDtoToEventType.apply(incidentDto.eventType));
    map.put("civils", incidentDto.civils);
    map.put("organisations", incidentDto.organisations);
    map.put("superbeings", incidentDto.superbeings);
    return map;
};
```

Figure 27 : Méthode IncidentDtoToMap

Ci-après, l'exemple de notre IncidentDto :

```
public class IncidentDto {  
    public Integer id;  
    public String ref;  
    public Date startDate;  
    public Date endDate;  
    public Boolean solved;  
    public Double dangerousness;  
    public String status;  
    public Boolean archive;  
    public EventTypeDto eventType;  
    public List<CommentDto> comments;  
    public List<Integer> civils;  
    public List<Integer> organisations;  
    public List<Integer> superbeings;
```

Figure 28 : Classe Incident DTO

Les Mapper sont des outils puissants mais qui demandent beaucoup de rigueur et de bien découper chaque étape de la transformation du type d'un objet.

#### 6.6.5 Présentation du front-end

Le rôle du front-end est donc de récupérer les informations envoyées depuis le Controller et de les afficher à l'utilisateur.

La première étape est de savoir si l'utilisateur a le droit de naviguer sur telle ou telle page. Cette information nous est fournie via le token de l'utilisateur. Si le token n'existe pas pour un utilisateur x, son accès à certaines pages sera refusé. L'idéal pour résoudre ce problème est de mettre en place un système pour faire en sorte qu'à la connexion d'un utilisateur l'application stocke le token qu'elle a reçu en réponse dans une variable temporaire. Nous utilisons ici le « localStorage » pour stocker cette donnée. Cette solution n'est pas idéale puisqu'elle comporte des failles de sécurité. En effet cette solution rend l'application vulnérable aux attaque XSS. Cependant ce n'est qu'une solution temporaire que nous améliorerons dans le futur. Il ne faut pas oublier que l'application que nous présentons n'est qu'un POC et donc requiert seulement une preuve des fonctionnalités basiques de celle-ci. Les éventuelles solutions que nous pourrions apporter pour sécuriser le stockage du token serait de par exemple couper le token en deux et de stocker une moitié dans le localStorage et l'autre moitié dans les cookies.

```
async handleSubmit() {  
    const response = await axios.post(`url: `/shield/login`, data: {  
        email: this.email,  
        password: this.password,  
    })  
    localStorage.setItem("token", response.data.token)  
    this.$router.push(Incident)
```

Figure 29 : Gestion du token d'authentification

Une fois authentifié et le token stocké dans le localStorage l'utilisateur peut se déplacer sur le site et accéder aux pages qui lui sont destinées. Pour se faire il est primordial de faire passer le token de l'utilisateur en paramètre de toutes requêtes qui va vers le serveur. Cette action est effectuée en ajoutant un tableau « `headers :{}` » à la requête avec une entrée « Authorization : 'Bearer ' + token ». Token que nous viendrons chercher dans le localStorage.

```
async loadIncidents() {
    await axios.get(url: "/shield/incident/", config: {
        headers: {
            "Access-Control-Allow-Origin": "*",
            "Access-Control-Allow-Credentials": true,
            Authorization: 'Bearer ' + localStorage.getItem(key: "token"),
        }
    })
}
```

Figure 30 : Renvoi du token au Controller

Par exemple une tentative d'accès à la page de liste des incidents lorsqu'un utilisateur ne possède pas encore de token afficherait ce résultat-là. Aucune donnée n'est autorisée à transiter.

The screenshot shows a dark-themed web application interface. At the top, there is a navigation bar with links: Home, Incidents, Litiges, Missions, and Login. Below the navigation bar, there is a button labeled "Créer un incident". The main content area has two sections: "Dangerosité" and "Résolu". Under "Dangerosité", it says "Attention : Jaune : Au dessus de 40%" and "Critique : Rouge : Au dessus de 70%". Under "Résolu", it says "Vert : Oui" and "Rouge : Non". Below these sections is a table header with columns: Référence, Timeframe de l'incident, Statut, Dangerosité, Type d'incident, Résolu, and Actions. The table body contains the message "Aucun incident pour l'instant".

Figure 31 : Liste incidents sans Token

En revanche dans le cas où un token est disponible le résultat sera le suivant :

The screenshot shows the same web application interface as Figure 31, but with data populated in the table. The table now lists several incidents with details such as reference numbers, dates, status, danger levels, types, and resolution status. Each row includes an "Actions" column with icons for editing and deleting.

Référence	Timeframe de l'incident	Statut	Dangerosité	Type d'incident	Résolu	Actions
I20210112095345	12/03/2021 -	À prendre en charge	70	Attaque de Civils	●	<span>i</span> <span>TO DO</span>
I20201225000645	12/03/2021 -	À prendre en charge	30	Cambriolage	●	<span>i</span> <span>TO DO</span>
TATAYOYO	17/03/2021 -	Traitemenent en cours	20	Attaque d'Ânumains et de Huânes	●	<span>i</span>
I20210112095332	26/03/2021 - 27/03/2021	Traitemenent en cours	60.67	Attaque de Huânes seulement	●	<span>i</span>
ISO7T6744I7654	25/03/2021 -	Intervention demandée	98.45	Attaque de Huânes seulement	●	<span>i</span>
I202101120485165	31/03/2021 -	Intervention demandée	41.21	Attaque de Huânes seulement	●	<span>i</span>
I20210112095332	25/03/2021 -	À prendre en charge	10	Attaque de Huânes seulement	●	<span>i</span> <span>TO DO</span>

Figure 32 : Liste incidents avec Token

Sources :

<https://github.com/swagger-api/swagger-core/wiki/Annotations>  
<https://www.baeldung.com/spring-boot-swagger-jwt>  
<http://springfox.github.io/springfox/docs/current/#configuring-startup>  
<https://www.baeldung.com/jpa-many-to-many>  
<https://www.codejava.net/spring-boot/security-role-based-authorization>  
<https://intellitech.pro/spring-security-how-to-auth-with-token>  
<http://blog.paumard.org/cours/jpa/chap05-heritage-single-table.html>  
<https://zetcode.com/all/#springboot>  
<https://stackabuse.com/a-guide-to-jpa-with-hibernate-relationship-mapping/>  
<https://dev.to/token-in-localstorage-vs-cookies>

## 6.7 Tests & recettage

### 6.7.1 Fiche de tests

Afin de pouvoir vérifier le comportement que nous désirons pour notre application, nous allons réaliser un test conditionné.

Pour ce faire, nous avons donc élaboré une Fiche de Test décrivant les actions à réaliser et les comportements attendus. Vous pouvez y accéder [ici](#).

Cette fiche nous permettra, en cas d'erreur de pouvoir faire remonter l'information et le contexte dans lequel intervient le bug à l'équipe de développement.

## Fiche de test fonctionnel de l'application S.H.I.E.L.D. R&D.

Ref :	Rédacteur : Sébastien XAVIER CARLOS	Testeur :	Date de Test : JJ/MM/AAAA			
Documents de références	Rapport.docx					
Étape(s) :	1. Connexion 2. Création 3. Consultation					
Cas de test :	1. Accéder au site  2. Se Connecter  3. Ajouter un Incident avec ses relations.  4. Consultation de la liste des incident  5. Affichage du détail d'un incident	Jeux de données :	1. url : http://localhost:3000/  2. Données de Connexion: "email": "joyce.flores@shield.incident.org" "password": "CryptedMdp1!"  3. Données incident: "ref": "AT20210406NY17" "startDate": "2021-04-06" "solved": 0 "archive": 0 "dangerousness": 92 "status": "Intervention demandée" "eventType": "name": "Attaque d'Ânumains" "civils": 1, 2 "organisations": 8 "superbeings": 2, 3 "comments": "comments": "Attaque d'un groupe d'Ânumains dirigés par Crâne Rouge" "comments": "Les Ânumains sont arriver sur terre suite à l'utilisation du tesseract par Crâne Rouge faisant entrer ces créatures d'un autre univers dans notre monde."			
Procédure de vérification de Conformité fonctionnelle		Résultat				
N°	Description	Résultat Attendu	OK	KO, Constat.	Ticket Bug	Niveau (Faible, Majeur, Bloquant)
1	Arriver sur le site	La page d'accueil s'affiche correctement.				
2	Clique sur l'onglet "Login"	La page de connexion s'ouvre.				
3	Entrée des données de connexion	Les champs du formulaire sont bien remplis avec le mdp caché.				
4	Cliquer sur "se connecter"	Arrivée sur la page d'incidents en étant connecté. Les informations des déclarations d'incident sont bien affiché et triés.				
5	Clique sur le bouton "créer un incident"	La page formulaire d'ajout d'incident s'affiche				
6	Entrée des données de création d'un nouvel incident	Les champs du formulaire sont bien remplis.				
7	Cliquer sur "créer un incident"	Un message "création réussi" apparaît. Redirection vers la page Incidents.				
8	Cliquer sur le bouton Information	La page affichant le détail de l'incident apparaît.				
9	Cliquer sur le bouton logout	Déconnexion et redirection sur la page home.				
Commentaires :						

À la suite de notre test, nous avons pu constater que le cas étudié était en adéquation avec nos attentes comportementales.

Si le comportement n'avait pas été celui attendu, nous aurions créé un ticket de bug afin de le décrire ainsi que tous les paramètres et les conditions dans lequel celui-ci est intervenu. Il est primordial de donner le cheminement complet des actions effectuées pour arriver au bug constaté. Cela permettra aux équipes de développement de pouvoir reproduire le bug et ainsi rechercher les origines de celui-ci.

De plus, l'ajout d'enregistrement des logs d'exceptions permettra la traçabilité des problèmes qui se présenteront à nous.

## 7 Manuel d'utilisation

Nous allons maintenant vous présenter notre Guide Utilisateur, il portera sur les fonctionnalités disponibles dans l'application que nous avons réalisées.

Le manuel est disponible [ici](#).

Il guidera les nouveaux utilisateurs et sera complété au fur et à mesure de l'avancement du développement des fonctionnalités de l'application ce qui les accompagnera dans l'évolution de leur travail.

The image displays two pages from the S.H.I.E.L.D. User Manual. The left page features the S.H.I.E.L.D. logo at the top, followed by the text "Manuel d'utilisation de l'application ©S.H.I.E.L.D." and "©MARVEL". A large, stylized red stamp reading "CONFIDENTIAL" is overlaid on the bottom half. The right page contains the text "Manuel d'utilisation de l'application ©S.H.I.E.L.D." and "©MARVEL" at the top, followed by "Mentions Légales". It includes the official seal of the United States Department of Justice, which features an eagle holding a shield, surrounded by the words "DEPARTMENT OF JUSTICE" and "UNITED STATES GOVERNMENT JUSTITIA SEQUITUR". Below the seal, the text "©S.H.I.E.L.D. Tous droits réservés." is visible. At the bottom of both pages, there is small print about the nature of S.H.I.E.L.D. as a private company. The footer of each page includes copyright information for 2021 and links to the table of contents.

## 8 Évolutions à venir

La conception et la réalisation d'un CRM étant conséquent, beaucoup d'évolutions sont encore à apporter à notre projet afin de pouvoir le mener à terme.

Nous allons vous présenter quelques idées que nous avons concernant l'évolution que pourra connaître notre projet :

- Terminer les différentes fonctionnalités identifiées dans la cartographie fonctionnelle, cette version représentera la V1.
- Mise en place du design prévu par la maquette.
- Mise en place de la traçabilité de toutes les actions effectuées sur l'application (Qui, Quand, Quoi, Où et Comment).
- Mise en place de l'historique des droits ainsi que l'externalisation des logs.
- Mise en place de l'ajout dynamique d'information permettant aux agents d'être instantanément informés.
- Mise en place d'une application smartphone pour les utilisateurs.
- Mise en place de la géolocalisation afin de permettre une intervention plus rapide et précise des agents.
- Mise en place de la reconnaissance faciale et digitale pour l'authentification.
- Amélioration de la sécurité de l'application côté front-end, notamment concernant le stockage du token).
- Mise en place du cryptage des mots de passe.
- Mise en place d'un générateur de mot de passe pour le JWT secret.
- Mise en place de serveur d'hébergement sécurisé (RAID5) avec du load balancing permettant ainsi la haute disponibilité de l'application.
- Mise en place de la gestion du multilangue
- Mise à disposition d'outils de synthèse et reporting
- Prévision et organisation des futures équipes de maintenance
- Extension de l'application aux autres planètes alliées

# 9 Conclusion

## 9.1 Rappel du contexte

Le S.H.I.E.L.D. souhaite mettre en place un CRM offrant de nombreuses fonctionnalités autour des thématiques suivantes :

- Déclaration d'incident
- Gestion des équipes de secours et des interventions de super-héros
- Surveillance à grande échelle de la population civile et des entités malveillantes
- Relations publiques

Au vu de la demande, les problématiques d'optimisation (code et base de données), de sécurisation des informations et de haute disponibilité pour les utilisateurs (disponibilité web) sont rapidement apparues comme critiques.

Elles ont donc naturellement orienté nos choix techniques tout au long de la phase de conception.

En tant qu'étudiants découvrant le langage Java, nous avions également des contraintes fortes :

- Les délais de livraison et plus particulièrement sur le jalon de développement
- Une montée en compétence sur les multiples sujets techniques abordés dans le projet

## 9.2 Conformité des réalisations

Ce projet nous parut assez simple à aborder au début. En effet après la relecture du projet nous devions envoyer un email à notre pilote de formation en lui expliquant quels modules du projet nous souhaitions réaliser. Après une première analyse du cahier des charges nous avions décidé de réaliser les modules suivants :

- Civil / Organisation
- Super-Héros / Super-Vilains
- Incident
- Mission
- Rapport de mission

Cependant, après avoir passé du temps sur ladite analyse et la conception de l'application nous nous sommes rendu compte de notre mauvaise estimation de temps. Nous avons alors décidé de mettre à jour nos objectifs pour cette fois ne réaliser que :

- Déclaration d'un incident par un civil
- Création d'un civil / super-héros
- Page de gestion des déclarations pour les employés
- Page de gestion des utilisateurs

Ces nouveaux objectifs n'ont pu être réalisé dans le temps qui nous était imparti, cependant nous restons fiers du résultat que nous avons fourni. La déclaration d'incident s'effectue désormais par un employé et non un civil. Les méthodes de création de Civil et Super-Héros sont présente au sein de l'application back mais n'ont pas été appliquée dans le front. La partie update concernant la gestion des déclarations d'incident est prête à 75% dans le back. En ce qui concerne la page de gestion des utilisateurs, cette dernière fonctionnalité est disponible dans le back aussi.

Concernant le respect des deadlines du projet établies à l'origine, celles-ci n'ont pas été respectées car nous avons du reprioriser nos objectifs et décaler certaines productions écrites afin de pouvoir solutionner les problématiques que nous rencontrions.

### 9.3 Problématiques rencontrées

Dans tous projets apparaissent des imprévus, que ce soit le fait de sous-estimé la masse de travail ou la difficulté de la tâche. L'important est de savoir rapidement réagir aux différentes problématiques qui se présentent à nous. Pour cela, un suivi d'avancement doit être mise en place et effectuée de façon rigoureuse. En effet, chaque semaine nous nous réunissions afin de faire un point sur la répartition des tâches ainsi que la résolution de problèmes qui se sont présentés aux membres de l'équipe.

Ainsi nous allons vous présenter les principaux problèmes qui se sont présentés à nous et les solutions que nous avons tenté d'y apporter :

- Définition et choix de notre architecture, n'ayant jamais pratiquer la mise en place d'une API from scratch nous avons dû grandement étudier les différents Design pattern afin de sélectionner celui qui convient le mieux à notre application. Après avoir fait appel à des professionnel du milieu qui nous ont aiguillé sur la démarche à suivre nous avons pu adopter un regard différent sur la stratégie à mettre en place afin de réaliser une API en java.
- Stratégie d'héritage entre les classes et leurs relations entre elles. Nous avons dû corriger certaines relations dans notre base de données.
- Nous avons dû utiliser des Mapper/Resource/DTO que nous ne connaissions pas jusque-là. Nous avons dû effectuer beaucoup de recherches et après avoir fait appel à des professionnels du métier nous avons pu monter en compétence afin de réaliser notre projet.
- La mise en place de la sécurité web en java nous était inconnue et a nécessité un certain temps d'étude.
- Nous avons rencontré un problème, avec l'IDE « Eclipse », lors des résolutions de conflit dans le cadre du versionnage. Nous avons par la suite décidé de changer d'IDE et d'utiliser « IntelliJ IDEA ».

## 9.4 Bonnes pratiques et axes d'améliorations

Lors de ce projet, nous avons mis en place certaines pratiques que nous estimons profitable à la réalisation d'un projet informatique professionnel. Nous allons vous les présenter ci-après :

- Nous avons amélioré la gestion de notre Trello ainsi que notre méthodologie de suivi de projet avec Gantt.
- Notre forte communication et l'implication de chacun ayant eu un comportement professionnel nous a permis de mener ce projet à bien.
- Notre analyse du besoin très poussé nous a permis de découper fonctionnellement nos tâches de développement et donc de gagner du temps sur la réalisation du code et éviter des temps de réappropriation et interrogation face au code.

Nous sommes satisfaits de la montée en compétence concernant la qualité du code.

Certains éléments de notre organisation nous ont paru peu concluants. Ainsi nous vous présentons ci-après les méthodologies que nous désirons ne plus reproduire et les pistes d'amélioration que nous souhaiterions apporter aux problématiques rencontrés :

- La mise en place du versionnage pourrait être mieux découpée. Nous avons effectué notre découpage par branches thématiques ; des ramifications fonctionnelles supplémentaires auraient été appréciées.
- Avec le recul sur ce projet, nous avons sous-estimé le temps que prenait l'analyse du besoin. Aujourd'hui nous prévoirions un temps de travail supplémentaire en début de projet.
- Pour respecter nos délais, nous avons volontairement omis la réalisation d'un backlog. Ce dernier nous aurait permis de suivre l'avancement du développement des fonctionnalités, de planifier des sprint et ainsi établir une roadmap. Pour ces mêmes raisons nous n'avons pas réalisé de user story. Celle-ci nous aurait permis de mieux définir l'UX des futurs utilisateurs, ce qui aurait grandement améliorer l'application, et de fixer des critères d'acceptation pour nos tests. De même, dans la continuité de la priorisation, nous n'avons pas réalisé de tests unitaires. Ces derniers auraient permis de garantir une meilleure stabilité et évolutivité du code.

## 9.5 Bilan individuel

Afin de pouvoir vous présenter le ressenti personnel de chaque membre du groupe, nous allons vous exposer un Retour sur Expérience (REX) individuel :

### ***Marie-Cécile PEYRAN :***

Ce projet était très intéressant, tant par l'originalité du sujet métier que par sa forte orientation conception. Il m'a permis d'appréhender la réalisation d'un projet informatique dans sa globalité.

Le fait de devoir estimer sa capacité de développement en découvrant l'environnement Java à constituer un beau challenge. Je pense que l'expérience a été formatrice pour la suite de mon parcours professionnel.

La réalisation des projets web/bdd et infra-réseau en parallèle m'a ralenti dans l'appréhension des technologies de ce projet. J'ai pu compenser cela grâce à la grande disponibilité de mes collègues. D'un point de vue organisationnel, la mise en place d'ateliers de code en groupe a permis de partager efficacement les informations au sein de l'équipe et d'optimiser nos temps de travail.

### ***Gabriel DESSERÉ-SUSINI :***

Ce projet fut très captivant d'autant par son épique thématique que l'équipe avec laquelle j'ai pu travailler, échanger et progresser. L'analyse du projet, bien que pesante, a été pour moi très intéressante et enrichissante. J'ai pu développer de nouvelles compétences dans le domaine UML et j'en suis ravi. Ayant eu un rejet du langage de programmation Java, en début d'année, par simple incompréhension de celui-ci, j'ai pu redécouvrir ce dernier cette fois-ci avec une approche différente. Ce qui m'a permis de le comprendre, et d'en acquérir de bonnes bases. Par ailleurs, être chef de projet accompagné de deux collègues expérimentés en la matière m'a permis de progresser grandement dans ce domaine. J'ai pu développer ma capacité d'analyse et de rédaction avec aisance et bonne humeur.

### ***Sébastien XAVIER CARLOS :***

Lors de ce projet j'ai pu réaliser mon premier projet en POO avancée. Cela m'a été grandement profitable et m'a appris énormément de choses que je n'avais alors pas intégré. J'ai pu ainsi comprendre le fonctionnement et l'utilité du design pattern MVC ainsi que ce que représente les DTO et les Resources. Je pense avoir beaucoup progresser en programmation lors de ce projet.

De plus j'ai pu développer mes compétences en conception d'application grâce à la mise en pratique de diagrammes UML. J'ai pu ainsi apprendre à transformer un besoin en spécifications fonctionnelles.

La complémentarité des compétences de notre groupe m'a également permis de m'imprégner des connaissances de chacun afin de renforcer les miennes.

En ce qui concerne les difficultés que j'ai rencontré, la première qui s'est présentée à moi fut la compréhension de l'architecture nécessaire à sa mise en place et son fonctionnement. Le fait de devoir se « jeter dans le grand bain » avec un cas concret à réaliser et sans tutoriel, m'a permis de me débloquer concernant la compréhension de la méthodologie à appliquer.

La seconde difficulté qui s'est présentée à moi fut l'assimilation du mécanisme des Mappers qui sont des concepts que je n'avais alors jamais rencontrés. Afin de m'aider dans ma compréhension j'ai fait appel à un développeur plus expérimenté que j'ai pu questionner sur le sujet. Le fait de mettre immédiatement en application ce que j'ai appris dans un sujet concret fut très profitable dans mon apprentissage.

Pour finir, j'ajouterais que la thématique Marvel parle à notre milieu professionnel souvent issu de la culture « geek ».

## 9.6 Bilan global

Dans un objectif d'amélioration des projets étudiant nous allons vous présenter notre retour concernant le projet en lui-même.

Tout d'abord, la thématique du projet nous a agréablement surpris et a été un facteur de motivation important qui nous a permis de mener à bien le projet. Le choix du CRM permet d'aborder de nombreuses problématiques métier que nous pourrions rencontrer dans notre milieu professionnel à l'avenir.

Nous avons trouvé le sujet clair et le cahier des charges était explicite.

La liberté de stratégie de développement fut très appréciable et nous permis de simuler un contexte professionnel.

La demande, du pilote de formation, concernant le fait de nous concentrer sur l'analyse du besoin et la conception, pendant les deux premiers mois, avant de commencer le développement fut très pertinente et nous permis de développer un nouveau regard quant à la conception d'une application. De plus cela nous a permis de développer nos compétences en UML.

L'opportunité de journées entières dédiées au projet lors de semaines de cours nous aida grandement.

Un point négatif du projet serait la période à laquelle les cours java ont été proposés. En effet il aurait été plus profitable d'y assister plus tôt dans l'année

## 9.7 Remerciements

Nous souhaitions remercier les personnes qui nous ont accompagné lors de ce projet et qui ont indirectement aider à l'accomplissement de celui-ci.

Monsieur Julien ROMEJON pour le partage de son savoir et des nombreux bons conseils qu'il nous a apporté dans l'architecture et la manière d'aborder le langage Java.

Monsieur Alexandre CLAIN pour sa disponibilité et son accompagnement tout au long du projet.

Monsieur Christophe SAURIN pour l'apport de connaissances et de nous avoir remis sur le bon chemin lors de la conception de notre application.

Madame Judith PIVOTEAU pour le temps qu'elle a su nous consacrer, sa pédagogie et son expertise Java.

## 9.8 Annexes

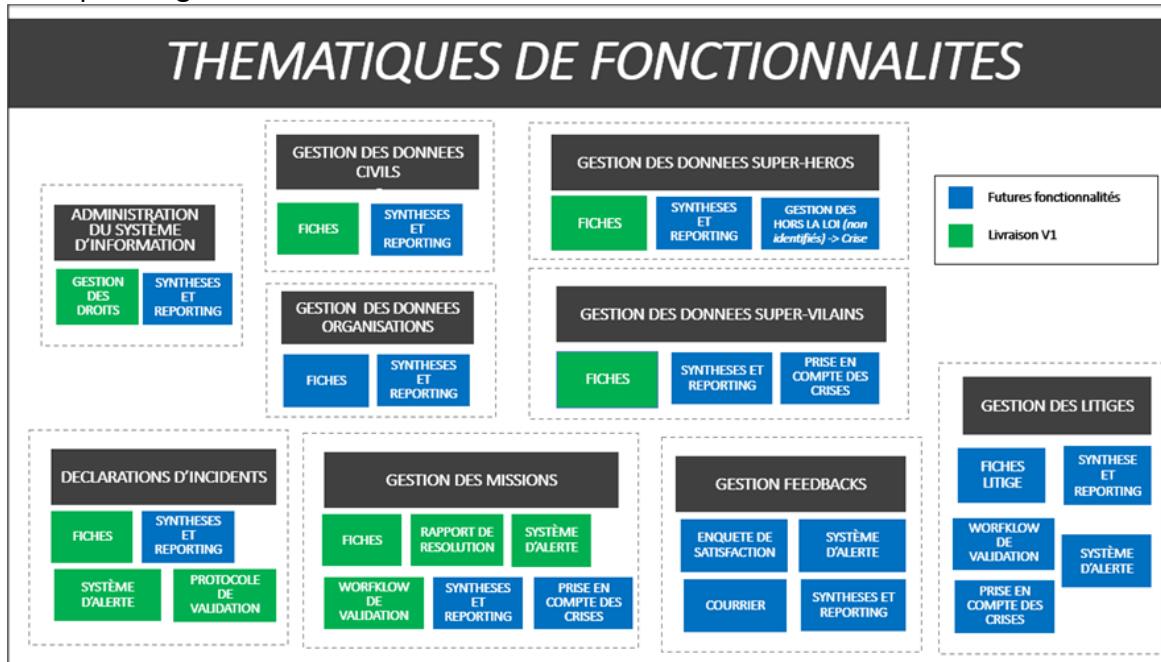
Vous pouvez retrouver notre projet sur le lien suivant :

<https://github.com/Kura33/ProjectJavaCesi>

Et le dossier des annexes [ici](#):

### 9.8.1 Première planification

Voici la première planification prévue avant analyse technique et l'échange de mails avec Alexandre Clain suite au rappel de l'attendu pédagogique et à la modification du planning de livraison.



## DEFINITION DES OBJECTIFS PEDAGOGIQUES DU PROJET JAVA

### Projet JAVA - Objectifs



CLAIN Alexandre  
Mer 02/03/2021 10:12

A : FERNANDES BASTIEN; MARCEL CLÉMENT; ROUBAIX CLÉMENT; DESSERE SUSINI GABRIEL; OBERLE GLENN; PEYRAN MARIE-CECILE +5 autres  
Cc : julien.romejon <julien.romejon@gmx.com>

Bonjour à tous,

Un petit mail pour que nous ayons tous le même discours et diminuer vos appréhensions sur le projet JAVA.  
Vous avez pu en parler avec Julien et nous échangeons actuellement sur le sujet aussi.

L'évaluation de ce projet JAVA porte principalement sur la conception et non le développement.  
Comme pour les autres projets à CESI, il sera demandé de justifier vos choix et cela aura plus d'importance que de le terminer.

A ce jour, il est effectivement compliqué d'ajouter des journées pour aborder JEE plus en profondeur ...  
La solution la plus pertinente est donc de faire un back en JAVA et un front en utilisant une technologie web.

Par contre, si vous souhaitez vous lancer un défi, n'hésitez pas !  
L'objectif de la formation, c'est de monter en compétences, pas d'obtenir des A ☺

Néanmoins, attention à l'utilisation de framework et de bien comprendre JAVA avant.  
Je préfère un petit POC pas terminé, mais bien codé, qu'une solution qui ne sera pas maintenable et que vous ne pourrez pas expliquer.

Cordialement,

## MODIFICATION DU PLANNING DE LIVRAISON

 CLAIN Alexandre <aclain@cesi.fr>  
Mar 16/02/2021 14:12  
À : XAVIER CARLOS SÉBASTIEN  
Cc : DESSERE SUSINI GABRIEL; PEYRAN MARIE-CECILE  
  
Bonjour à tous,  
  
C'est ok pour moi !  
Vous pouvez ajouté ce mail dans vos annexes.  
  
Cordialement,

 XAVIER CARLOS SÉBASTIEN  
Dim 14/02/2021 13:49  
À : CLAIN Alexandre <aclain@cesi.fr>  
Cc : DESSERE SUSINI GABRIEL; PEYRAN MARIE-CECILE  
  
Bonjour Alexandre,  
Nous revenons vers toi aujourd'hui afin de te soumettre à validation les nouveaux objectifs que nous pensons atteindre. Suite à l'analyse complète du besoin que nous avons effectué et le temps de travail que nous avons de disponible nous avons pu constater que les objectifs que nous t'avions présentés précédemment n'étaient pas réalisables. Ainsi nous te soumettons une nouvelle proposition de réalisation.  
Tout d'abord nous souhaitons réaliser un site web afin que l'application soit aisément disponible à tous les utilisateurs. Au vu des délais nous nous sommes recentrés sur les fonctionnalités essentielles à l'application. Ainsi les différents modules que nous pensons pouvoir réaliser sont :  
  
déclaration d'incident par un civil  
  
création de civil / héros  
  
e page de gestion des déclarations pour les employés  
Une page de gestion des utilisateurs  
  
Dans l'attente de ton retour,  
Cordialement,  
  
Marie-Cécile PEYRAN  
Gabriel DESSERE-SUSINI  
Sébastien XAVIER CARLOS  
  
S.H.I.E.L.D. R&D

## 1<sup>e</sup> PROPOSITION DE LIVRAISON

 CLAIN Alexandre <aclain@cesi.fr>  
Jeu 17/12/2020 16:04  
À : XAVIER CARLOS SÉBASTIEN  
Cc : DESSERE SUSINI GABRIEL; PEYRAN MARIE-CECILE  
  
Sébastien,  
  
Bien noté ! Pensez à être capable d'expliquer oralement comment vous auriez pu faire le reste.  
Vous pouvez ajouter ce mail à votre livrable final.  
  
Cordialement,



XAVIER CARLOS SÉBASTIEN  
 Jeu 17/12/2020 15:37  
 A : CLAIN Alexandre <aclain@cesi.fr>  
 Ce : DESSERE SUSINI GABRIEL; PEYRAN MARIE-CECILE



Bonjour Alexandre,  
 Nous te communiquons nos engagements concernant l'avancement de développement des modules de l'application.  
 Nous estimons pouvoir à minima développer les modules suivants :

Civils  
 Super-Héros  
 Super-Vilains  
 Incident  
 Missions  
 Rapports de Missions

Nous avons choisi ces modules car ils nous paraissent essentiels au fonctionnement de l'application.  
 Selon l'avancement de notre développement et suite aux points réguliers que nous effectuerons, nous pourrons être amenés à te proposer une modification de cette liste.  
 Nous te fournirons rapidement un visuel de la conception de notre application afin de pouvoir juger de la cohérence et de la faisabilité de cette dernière.

Cordialement

Marie-Cécile PEYRAN  
 Gabriel DESSERE-SUSINI  
 Sébastien XAVIER CARLOS

S.H.I.E.L.D. R&D

### 8.8.2 Personas

## *[Inconnu], Super-vilain*



**CONTEXTE**

Sof de vengeance suite à un passé douloureux.

Son but est de nuire aux Avengers par n'importe quel moyen et/ou de s'enrichir.

**BUT / COMPORTEMENTS :**

Rédige des menaces

Crée de « faux » incidents dans l'application pour faire perdre du temps au S.H.I.E.L.D.

Essaie de nuire à l'intégrité des données (falsification, suppression)

Essaie de voler les données du S.H.I.E.L.D.

**COMMENT LE CONTRER ?**

*On veut l'empêcher de mener à bien ses plans machiavéliques.*

Dans le cas où la demande est faite par un civil :

- Vérification de l'identité des civils déclarant un incident (vérifie en base qu'il ne s'agit pas d'un super-vilain) + le cas échéant, localisation de l'expéditeur
- Protocole de validation de la demande d'intervention par un membre de S.H.I.E.L.D.

Sécurisation de la base de données

[Retourner à la table des matières](#)

54

# Gerald Applesauce III, Civil



## CONTEXTE

Civil lambda

C'est un grand fan de Spider-Man.  
Se range inconditionnellement au côté de son héro préféré. Cela a même une incidence sur son orientation politique.

## BUT / COMPORTEMENTS :

- Constate les méfaits d'un super-vilain mettant un danger la vie ou la sécurité d'autrui et le rapporte aux autorités compétentes
- Poste des photos de son Avenger préféré sur les réseaux sociaux
- Envoie chaque année une carte postale à Spider-Man.
- Remplit avec assiduité le sondage en ligne après résolution d'un incident.
- En cas de problème lors de l'intervention, renseigne un formulaire de litige

## QU'EST-CE QU'ON VEUT LUI APPORTER ?

- Possibilité de déclarer l'incident constaté
- Les différentes fonctionnalités à sa disposition doivent lui apporter des sentiments positifs : appartenance, implication et l'impression d'être en sécurité.
- Possibilité de s'exprimer en cas de problème : que ce soit à travers la déclaration d'incident mais également le formulaire de litige.