Guillaume WAROUX

RAPPORT DE STAGE
pour le
Titre Professionnel « Concepteur Développeur d'Applications »

Sesame it

Cyber-sécurité | Sondes de détection

96 Bis Boulevard Raspail 75006 Paris

Sommaire

Remerciements

Je souhaite dans un premier temps remercier le Greta de Compiègne qui m’a permis de mettre en œuvre mon projet de reconversion professionnelle, ainsi que les différents formateurs de m’avoir transmis leurs connaissances et expériences.

Je remercie vivement la startup Sesame IT qui m’a offert l’opportunité d’appliquer les différentes connaissances au sein de leur entreprise, ainsi que l’accompagnement technique et humain dont chaque salarié à fait preuve à mon égard.

Présentation de l’entreprise

Sesame IT est une start-up créée en 2017 spécialisée dans la cybersécurité et située dans le 6ème arrondissement de Paris. Elle est composée de Audrey GAYNO-AMEDRO (CEO), Jérôme GOUY (CTO), Maud DELAUNAY (Lead developer), Jean-Philippe JUBENOT (Architecte système et réseaux), et Philippe (Développeur).

Elle proposera une sonde de détection (IDS) dans le cadre de la LPM (Loi de Programmation militaire) pour les OIV (Opérateurs d’Importance Vitale).

Résumé

J’ai rejoins l’équipe de Sesame IT en tant que stagiaire développeur web.

La sonde « Jizo » est une sonde de détection for l’analyse de réseaux et l’analyse de fichier. Cette sonde peut-être gérée via un terminal, ou via une application qui est en cours de développement. Mon travail est de contribuer à la conception et au développement de cette application.

L’application permettra la gestion de la sonde mais aussi de voir l’état du hardware, un dashboard avec les dernières alertes, les métadonnées, les logs montrant les changements sur la sonde, et une analyse graphique.

Mon stage est divisé en deux périodes, la première période est du 23 Septembre 2019 au 25 Octobre 2019. La deuxième période est du 6 Janvier 2020 au 21 Février 2020.

Dans la première période de mon stage, j’ai appris l’environnement de développement, et à utiliser les processus internes. Je me suis adapté assez rapidement, et j’ai pu participer au développement en cours des différentes fonctionnalités.

Mon projet est apparu dans la seconde période de mon stage.

Le projet est de créer et développer l’extraction de fichiers de la sonde avec l’application. Il est supervisé par le chef technique, Jérôme GOUY.

Pour mener ce projet, j’ai du m’adapter aux technologies existantes.
Concrètement, il existe deux applications qui communiquent entre-elles.
La première, l’application PHP, utilise le framework Symfony pour créer les vues et communiquer avec la seconde application.
La seconde, l’application JAVA, utilise le framework Spring pour communiquer avec les bases de données.

Après mon projet, j’ai vécu une réelle expérience enrichissante. J’ai appris avec les problèmes rencontrés. J’ai travaillé avec des collègues toujours disponibles pour m’aider. Maintenant, je connais les différentes étapes pour

initier, créer, développer et intégrer un projet très intéressant dans un domaine qu'est la cybersécurité.

Project Summary

I joined the IT team of Sesame IT as a web developer intern.
The prob »Jizo » is a detection probe for the analysis of networks and analysis of files. This probe can be used by the terminal or by an application which is still under development.

My job was to contribute to the conception and the development of this application.

This application will help manage the probe but also see the probe's hardware, a dashboard with the latest alerts, the metadata, the logs showing the changes on the probe and a graphic analysis of the information.

My internship was divided into two periods. The first period was from the 23rd of September 2019 to the 25th of October 2019 and the second period started the 6th of January 2020 and ended the 21st of February 2020.

In the first period of my internship, I learnt how to develop in a specific environment and how to use an intern process. I adapted quite easily, therefore I could participate in the ongoing development of the different features.
My project began to emerge in the second part of my internship.

The project was to create and to develop the extraction of the files from the probe via the application. It was supervised by Mr Jérome GOUY, the Chief Technical Officer.(CTO)
To carry out this project, I had to adapt to the current technologies.
In practical terms, two applications communicate with each other.
The first one : the PHP application uses the framework Symfony to create views and communicate with the second application. The second one : the JAVA application uses the framework Spring to communicate with the databases.

At the end of my project, I realized that it was a great and rewarding experience. I learnt a great deal by solving several problems. I worked with colleagues who were really helpful. Now I know the various steps necessary to design, create and develop a very interesting project in the field of cybersecurity.

Compétences couvertes par le projet		
Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité	1. Maquetter une application	X
	2. Développer un interface utilisateur de type desktop	
	3. Développer des composants d'accès aux données	X
	4. Développer la partie front-end d'une interface utilisateur web	X
	5. Développer la partie back-end d'une interface utilisateur web	X
Concevoir et développer la persistance des données en intégrant les recommandations de sécurité	6. Concevoir une base de données	
	7. Mettre en place une base de données	X
	8. Développer des composants dans le langage d'une base de données	X
Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité	9. Collaborer à la gestion d'un projet informatique et l'organisation de l'environnement de développement	X
	10. Concevoir une application	X
	11. Développer des composants métier	X
	12. Construire une application organisée en couches	X
	13. Développer une application mobile	
	14. Préparer et exécuter les plans de tests de l'application	
	15. Préparer et exécuter le déploiement de l'application	

Partie 1 – DESCRIPTION ET CAHIER DES CHARGES DU PROJET

La sonde « Jizô » est une sonde de détection qui permettra d'analyser les flux réseaux, d'extraire des fichiers et d'analyser les fichiers. Cette sonde peut être gérée directement via une console, ou via une application qui est en cours de développement. Mon rôle est de participer activement à la conception et au développement de cette application.

L'application permettra de retrouver la gestion de la sonde mais aussi de voir un état du hardware de la sonde, un dashboard remontant les dernières alertes, les métadonnées, les logs indiquant les changements sur la sonde, et une analyse graphique des informations.

Le projet est de concevoir et développer la partie d'extraction de fichiers de la sonde via l'application.

Le projet est mené par Jérôme GOUY, Maud DELAUNAY, et moi-même.

Pour concevoir et développer cette partie, il était indispensable de s'adapter aux technologies déjà mises en place. L'application partiellement développée a suivi les indications de l'ANSSI pour le développement, ce qui justifie l'architecture déjà mise en place. Techniquement, deux applications communiquent entre-elles pour le fonctionnement de l'application.

Analyse de l'existant

Aujourd'hui la sonde de détection est en cours de développement, et commence les différentes certifications auprès de l'ANSSI.

La sonde propose actuellement :

- L'analyse de flux réseau.
- L'analyse de fichier.
- Une application web qui permet :
  - Dashboard, permettant de voir l'état du hardware et les différences interfaces.
  - Une page de configuration de la sonde.
  - La gestion des règles.
  - Les remontées d'alertes.
  - L'affichage des métadonnées et des fichiers.
  - L'affichage des logs et des différents graphiques.

Les utilisateurs du projet

La sonde de détection est développée dans le cadre de la Loi de Programmation Militaire pour les OIV(Opérateurs d'importance vitale). Ce contexte implique à la sonde qu'elle soit hermétique à toutes connexions extérieures au réseau de la cible.

Par ce fait, une fois la sonde de détection déployée, seuls les utilisateurs prévus par la cible peuvent la manipuler via l'application web.

Les échéances du projet

La durée du projet est estimée à environ 3 semaines.

Le projet a commencé le lundi 27 Janvier 2020, jusqu'au 14 Février 2020, mais il s'agit d'une cadre pour le rythme de développement. Cette date de fin n'est donc qu'indicative, et permet de suivre un rythme fixé par Jérôme GOUY.

Contexte technique

La sonde de détection est un serveur physique déployée dans l'infrastructure du client, et donc accessible à distance via l'application web partiellement développée.

L'application devra donc continuer à fonctionner sur les 2 principaux navigateurs :

- Mozilla Firefox
- Google Chrome

Le développement de l'application a commencé dans un cadre particulier imposé par l'ANSSI. Précisément, deux applications communiquent entre-elles. Une application PHP gérée par le framework Symfony va s'occuper de communiquer avec l'autre application, et de générer les vues, contrôler les informations saisies par l'utilisateur, etc. La deuxième application JAVA gérée par le framework Spring va s'occuper de recevoir les informations de l'application PHP, et de communiquer avec la base de données.

En l'occurrence, la base de données utilisée est MariaDb.

Définition des entités

Concernant le projet, les entités sont et seront :

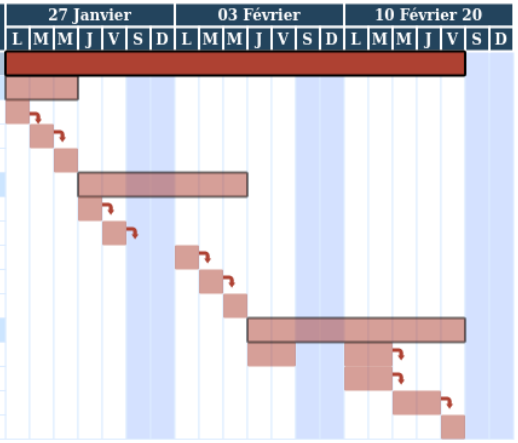
- **La sonde,** qui est une entité déjà existante**,** permet de gérer les différentes fonctionnalités principales, telles que l'affichage du dashboard, des alertes, des logs, des graphiques et la gestion des différentes règles.
- **Le hash** qui sera rattaché à l'entité Sonde. La sonde pourra contenir plusieurs hashs.

Partie 2 – GESTION DE PROJET

Le projet est mené par 3 personnes dont moi-même, où Jérôme GOUY assumera surtout la supervision du projet. Je m'occuperai de la conception du projet en étroite collaboration avec Jérôme, ainsi que le développement sous l'encadrement de Maud DELAUNAY.

Pour plus de détails, j'ai réalisé un diagramme de GANTT :

	Nom de la tâche	Durée	Début	Fin
1	Projet complet	13 jours	27.01.20	14.02.20
2	Définition et analyse du projet	3 jours	27.01.20	29.01.20
3	Définition du projet	1 jour	27.01.20	27.01.20
4	Analyse et définition des besoins fonctionnels et non fonct	1 jour	28.01.20	28.01.20
5	Réunion d'équipe du projet (organisation, ...)	1 jour	29.01.20	29.01.20
6	Conception du projet	5 jours	30.01.20	05.02.20
7	Conception générale du projet (DUC, Diagramme domain	1 jour	30.01.20	30.01.20
8	Elaboration des cas d'utilisation détaillés	1 jour	31.01.20	31.01.20
9	Maquettage des IHMs	1 jour	03.02.20	03.02.20
10	Elaboration des diagrammes (package, séquence,...)	1 jour	04.02.20	04.02.20
11	Réunion d'équipe pour validation (conception, ...)	1 jour	05.02.20	05.02.20
12	Développement du projet	9 jours	06.02.20	14.02.20
13	Codage des différentes classes (Controllers, Entités,...)	4 jours	06.02.20	11.02.20
14	Codage des templates	2 jours	10.02.20	11.02.20
15	Tests et déploiement	2 jours	12.02.20	13.02.20
16	Réunion d'équipe (bilan du projet, prochains développeme	1 jour	14.02.20	14.02.20



Partie 3 - ANALYSE FONCTIONNELLE

Les acteurs

La sonde détection est manipulable par 3 acteurs différents.

Admin

L'admin n'a aucune visibilité sur la gestion des hashs, il assume surtout la gestion des utilisateurs de la sonde, ainsi que la configuration de certaines parties du moteur de détection.

Operator

L'operator est l'acteur ayant accès au plus grand nombre de fonctionnalités de la sonde. Concernant le projet, il pourra ajouter, modifier, supprimer et afficher les différents hashs présents sur la sonde.

Auditor

L'auditor est l'acteur ayant le moins de privilèges au sein de l'application, pouvant juste consulter les règles, et les hashs.

Les besoins fonctionnels

Pour le cas d'utilisation « Gérer Hash IDS/DPI », plusieurs spécifications fonctionnelles et non-fonctionnelles après l'analyse du cahier des charges.

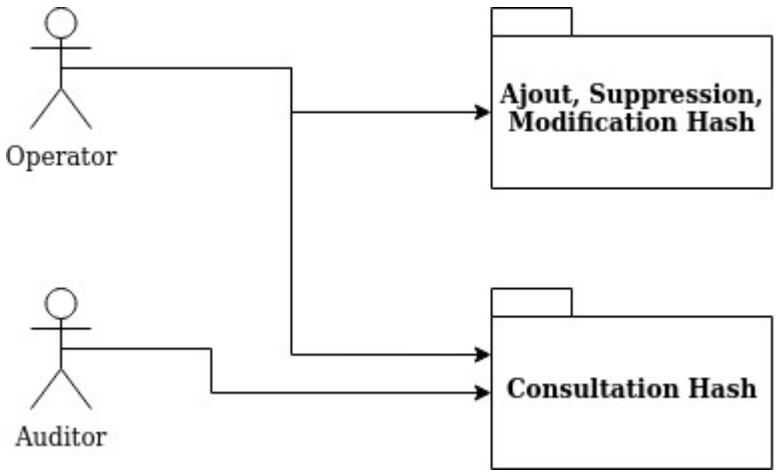
Les besoins fonctionnels sont :

- Le module doit respecter l'architecture déjà mise en place.
- L'ajout d'un hash ou de hashs devra se faire via un fichier que l'utilisateur doit téléverser.
- Les hashs pourrons être ajoutés selon des algorithmes prédéfinis.
- Un hash pourra être édité, mais les informations essentielles ne pourront pas être modifées.
- Un hash ou plusieurs hashs pourront être supprimés de la sonde.
- L'affichage des hashs pourra être filtré.
- Les hashs devront être stockés dans la base de données.
- La communication entre le back-end et la base de données se fera via des procédures stockées comme ce qui est déjà utilisé pour les autres modules de l'application.

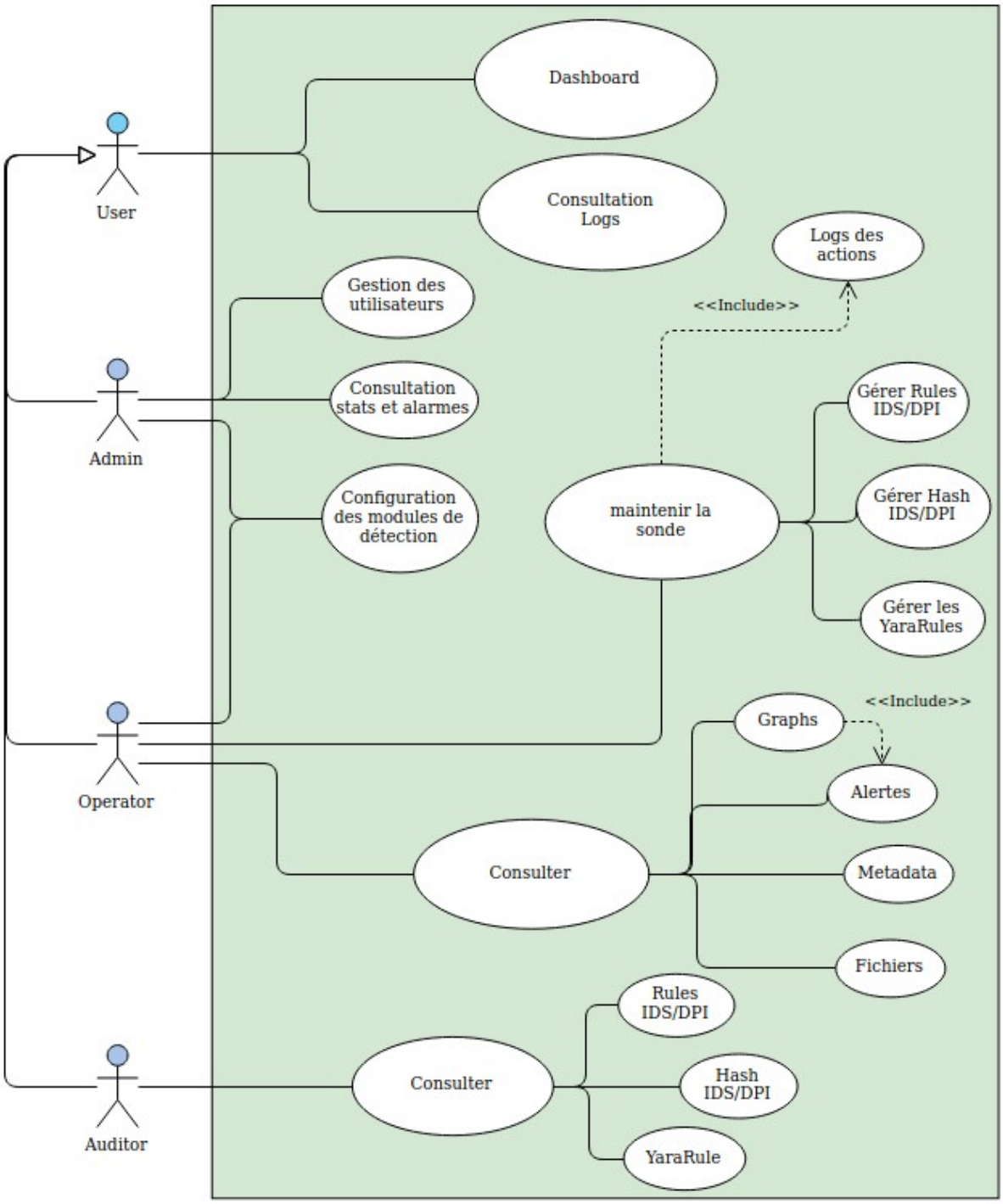
Les besoins non-fonctionnels sont :

- Un seul rôle peut gérer les hashs, les autres rôles pourront seulement les consulter, et voir l'évolution des hashs présents sur la sonde.
- Correspondre à la charte graphique déjà présente sur le reste de l'application.

Afin de simplifier la compréhension des acteurs avec les différents besoins, voici un diagramme plus explicite :



Cas d'utilisation



Le cas d'utilisation « Gérer Hash IDS/DPI » comprend un CRUD (Create, Read, Update, Delete).

Cas d'utilisation détaillés

Gérer Hash IDS/DPI > Ajouter Hash

Résumé : L'utilisateur ajoute un Hash à la sonde de détection.

Acteur(s) : Operator.

Précondition(s) : L'utilisateur est identifié avec le rôle « Operator » et se trouve sur la page « Hash IDS/DPI ».

Scénario Nominal

Étape	Description	Alt/Exc
1.	L'utilisateur sélectionne l'explorateur de fichier via le bouton « Browse »	Alt A
2.	Le système renvoie la fenêtre de l'explorateur de fichier	
3.	L'utilisateur sélectionne le fichier souhaité et valide	
4.	Le système enregistre le fichier sélectionné et ferme la fenêtre de l'explorateur de fichier	
5.	L'utilisateur choisit son algorithme de hachage, ajoute éventuellement un commentaire, et téléverse le fichier	
6.	Le système ajoute le fichier, rafraîchit la page et notifie l'utilisateur de l'ajout du fichier	Exc A, B, C

Postcondition : Le Hash est ajouté à la sonde de détection

Scénario(s) Alternatif(s)

Titre : Tentative d'ajout d'un Hash sans avoir sélectionné de fichier au préalable **= altA**

Étape	Description
1.A	L'utilisateur sélectionne le bouton « upload » sans avoir sélectionné de fichier au préalable
2.A	Le système ne recharge pas la page et envoie une notification à l'utilisateur de sélectionner un fichier

Postcondition : Le Hash n'est pas ajouté

Scénario(s) d'exception

Titre : Extension invalide = **ExcA**

Étape	Description
1.E	Le système refuse l'ajout du fichier car l'extension n'est pas celle souhaitée, et notifie à l'utilisateur la raison de l'échec d'ajout

Postcondition : Le Hash n'est pas ajouté

Titre : Le fichier importé ne correspond pas à l'algorithme choisi **= excB**

Étape	Description
1.E	Le système n'ajoute pas le fichier car l'algorithme de hashage est différent et notifie les raisons de l'échec de l'ajout à l'utilisateur

Postcondition : Le Hash n'est pas ajouté

Titre : Le contenu du fichier est non recevable **= excC**

Étape	Description
1.E	Le système refuse l'ajout du fichier car le contenu n'est pas recevable par le système, et notifie l'utilisateur

Postcondition : Le Hash n'est pas ajouté

Gérer Hash IDS/DPI > Modifier Hash

Résumé : L'utilisateur un Hash de la sonde de détection.

Acteur(s) : Operator.

Précondition(s) : L'utilisateur est identifié possédant le rôle « Operator » et se trouve sur la page « Hash IDS/DPI ».

Scénario Nominal

Étape	Description	Alt/Exc
1.	L'utilisateur sélectionne le hash à modifier via le bouton « edit »	
2.	Le système renvoie une page avec les informations du hash dont l'attribut « commentaire » est modifiable	
3.	L'utilisateur modifie le commentaire et valide	Alt A
4.	Le système modifie l'attribut « commentaire » du fichier hash, redirige sur la page de gestion des hash et notifie l'utilisateur de la modification faite avec succès	

Postcondition(s) : Le Hash est modifié de la sonde de détection

Scénario(s) Alternatif(s)

Titre : L'utilisateur annule la modification du hash **= altA**

Étape	Description
1.A	L'utilisateur annule la modification du hash en sélectionnant le bouton « Return to list »
2.A	Le système renvoie l'utilisateur sur la page de gestion des hash

Postcondition(s) : Le hash n'est pas modifié

Gérer Hash IDS/DPI > Supprimer Hash

Résumé : L'utilisateur supprime un Hash de la sonde de détection.

Acteur(s) : Operator.

Précondition(s) : L'utilisateur est identifié possédant le rôle « Operator » et se trouve sur la page « Hash IDS/DPI ».

Scénario Nominal

Étape	Description	Alt/Exc
1.	L'utilisateur sélectionne le hash à supprimer via le bouton de type « CheckBox » associé au hash	Alt A Exc A
2.	Le système demande via une fenêtre la confirmation de suppression à l'utilisateur	
3.	L'utilisateur valide la suppression	
4.	Le système supprime le hash selectionné, rafraîchit la page de gestion et notifie l'utilisateur du succès de la suppression et notifie l'utilisateur de la modification faite avec succès	

Postcondition(s) : Le hash est supprimé de la sonde de détection

Scénario(s) Alternatif(s)

Titre : L'utilisateur sélectionne plusieurs hash à supprimer **= altA**

Étape	Description
1.A	L'utilisateur sélectionne les hash qu'il souhaite supprimer via le bouton de type « CheckBox » associé au hash
2.A	Le système demande via une fenêtre la confirmation de suppression à l'utilisateur
3.A	L'utilisateur valide la suppression
4.A	Le système supprime les hash selectionnés, rafraîchit la page de gestion et notifie l'utilisateur du succès de la suppression

Postcondition(s) : Les hashs sont supprimés de la sonde de détection

Scénario(s) d'exception

Titre : Aucun hash n'a été sélectionné **= excA**

Étape	Description
1.E	Le système renvoie à l'utilisateur qu'aucun hash n'a été sélectionné

Postcondition : Aucun hash n'est pas supprimé

Gérer Hash IDS/DPI > Afficher Hash

Résumé : L'utilisateur affiche les hashs de la sonde de détection.

Acteur(s) : Operator.

Précondition(s) : L'utilisateur est identifié possédant le rôle « Operator » et se trouve sur n'importe quelle page de la sonde de détection.

Scénario Nominal

Étape	Description	Alt/Exc
1.	L'utilisateur sélectionne l'onglet « Rules »	
2.	Le système affiche les pages affiliées à l'onglet via une liste déroulante de type « DropDown »	
3.	L'utilisateur sélectionne la page « Hash IDS/DPI »	Alt A
4.	Le système renvoie la page de gestion des hash qui est composé de la liste des hash	

Postcondition(s) : Les hashs sont affichés à l'utilisateur

Scénario(s) Alternatif(s)

Titre : Affichage des hash correspondant aux filtres appliqués = **altA**

Étape	Description	Alt/Exc
1.A	L'utilisateur sélectionne le ou les filtres souhaités	
2.A	Le système renvoie la page de gestion des hash avec la liste correpondante selon les filtres	Exc A/Exc B

Postcondition(s) : L'utilisateur voit le(s) hashs correspondant(s) au(x) filtre(s)

Scénario(s) d’exception

Titre : Le filtre de date de départ est postérieure au filtre de la date de limite = **excA**

Étape	Description
1.E	L’utilisateur sélectionne le ou les filtres dates avec les valeurs souhaitées
2.E	Le système renvoie une notification pour avertir que les valeurs sont incorrectes

Postcondition(s) : L’utilisateur est sur la page de liste des hashs

Titre : Aucun hash ne correspond aux filtres appliqués = **excB**

Étape	Description
1.E	L’utilisateur sélectionne le ou les filtres souhaités
2.E	Le système renvoie une liste vide notifiant à l’utilisateur qu’aucun résultat n’a été trouvé

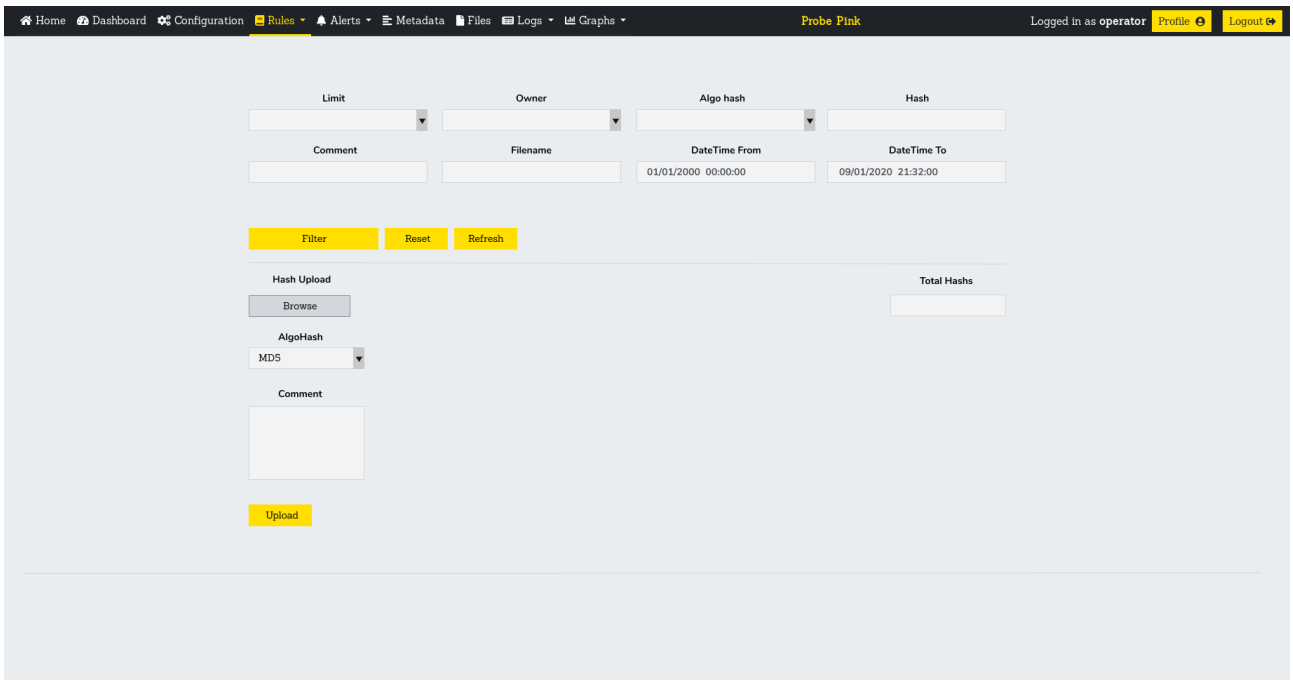
Postcondition(s) : L’utilisateur voit une page lui indiquant aucun résultat

Maquettage

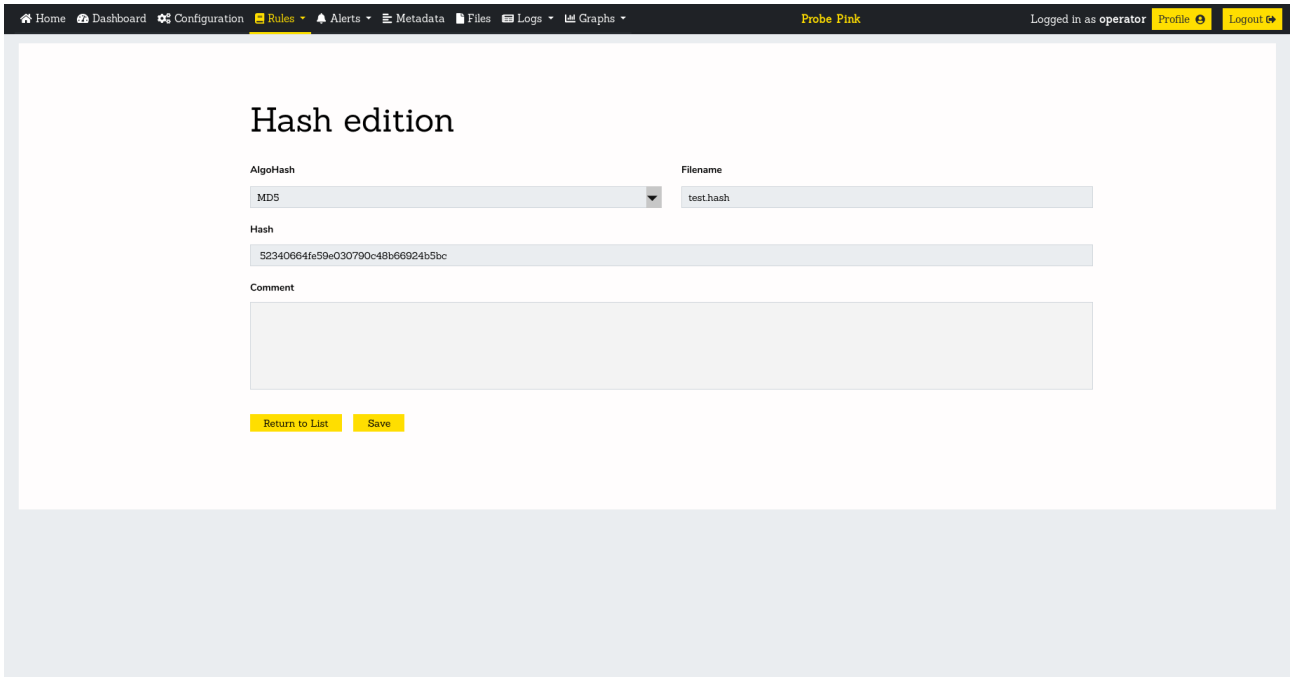
Les maquettes ont été faites via l’outil Figma, disponible en ligne. Pour effectuer les différentes maquettes, j’ai considéré la charte graphique ainsi que les différentes structures de pages qui ont déjà été développées.

Grâce à ces informations précises, j’ai pu faire des maquettes précises.

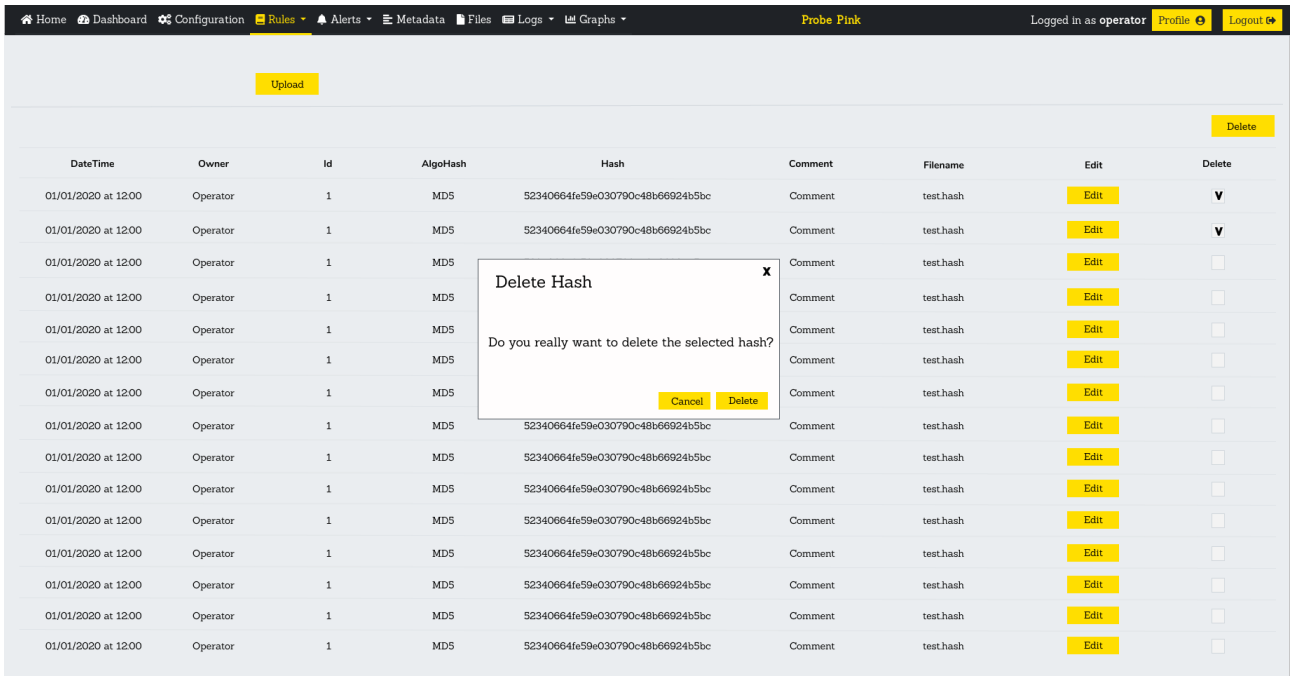
Maquette « Gérer hash IDS/DPI » > Ajouter hash :



Maquette « Gérer hash IDS/DPI » > Modifier hash :



Maquette « Gérer hash IDS/DPI » > Supprimer hash :



Maquette « Gérer hash IDS/DPI » > Afficher hash :

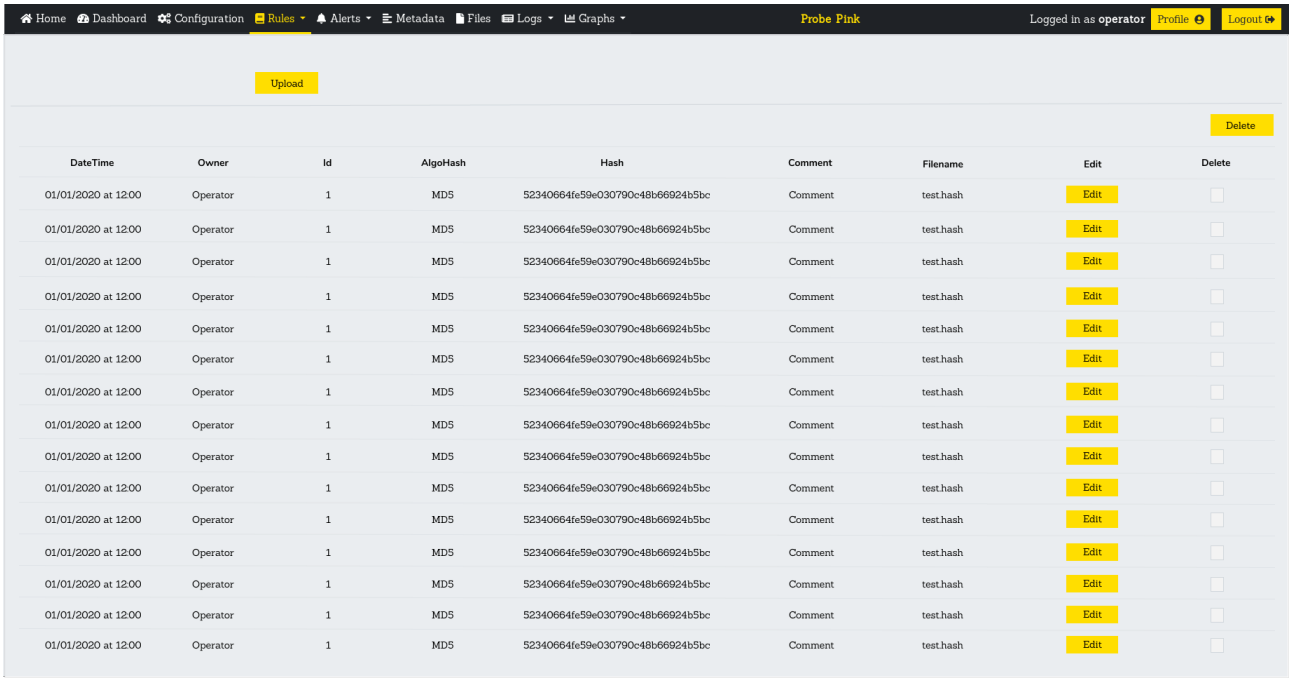
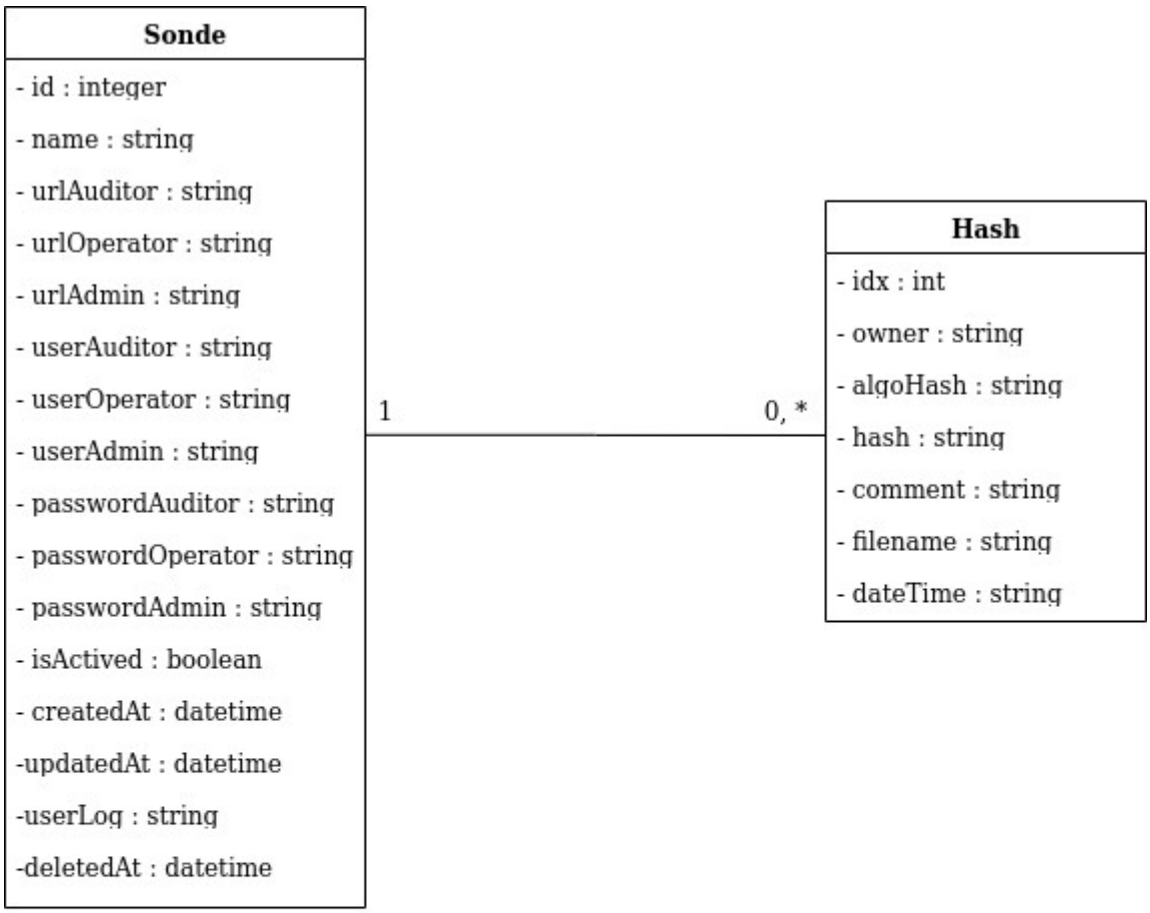


Diagramme de classes d'analyse du projet

Après analyse, 2 classes ressortent pour le développement de ce projet.

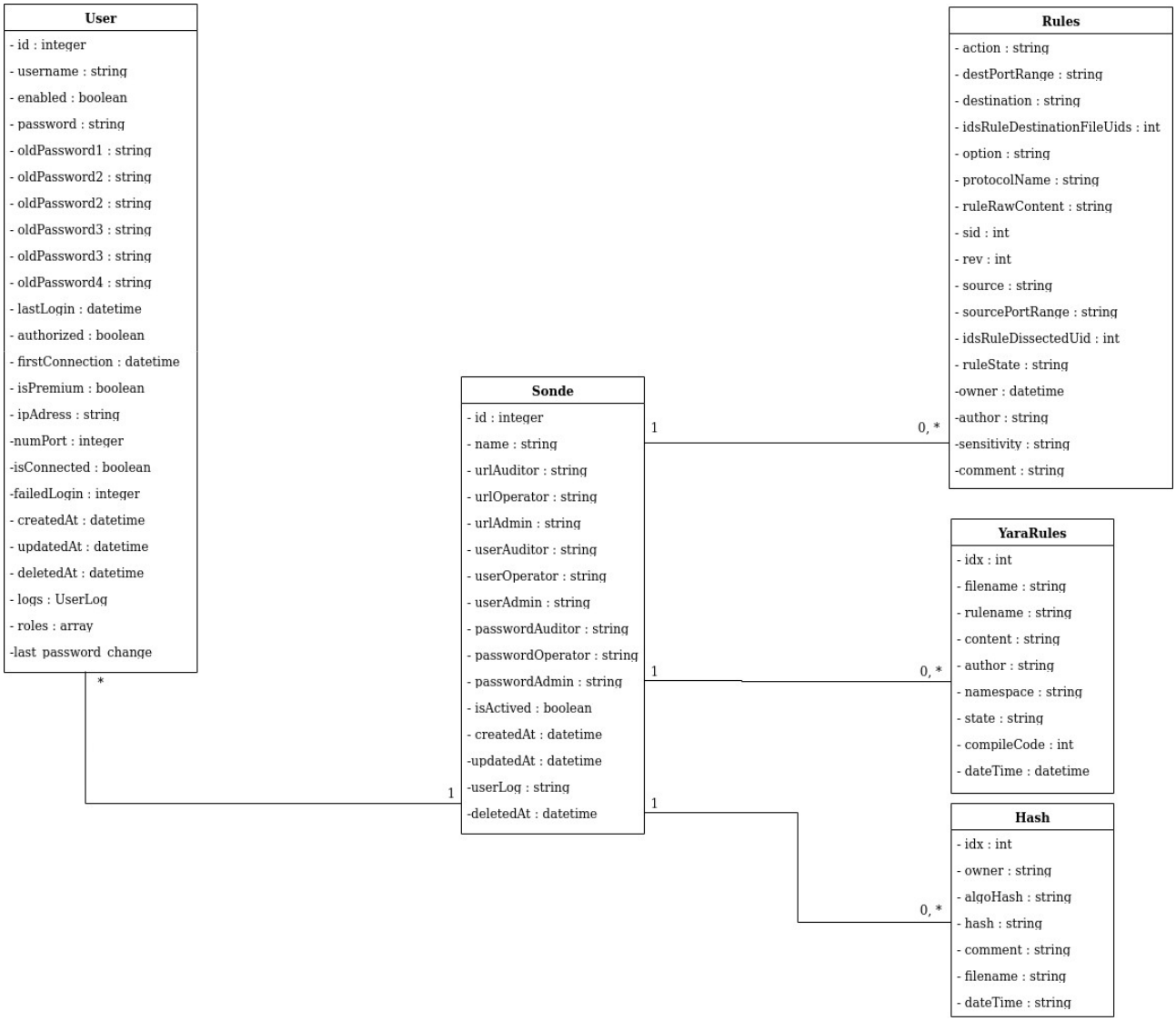
Une sonde pourra contenir aucun hash, ou plusieurs. En revanche, un hash ne pourra appartenir qu'à la sonde qui l'a ajouté.



*

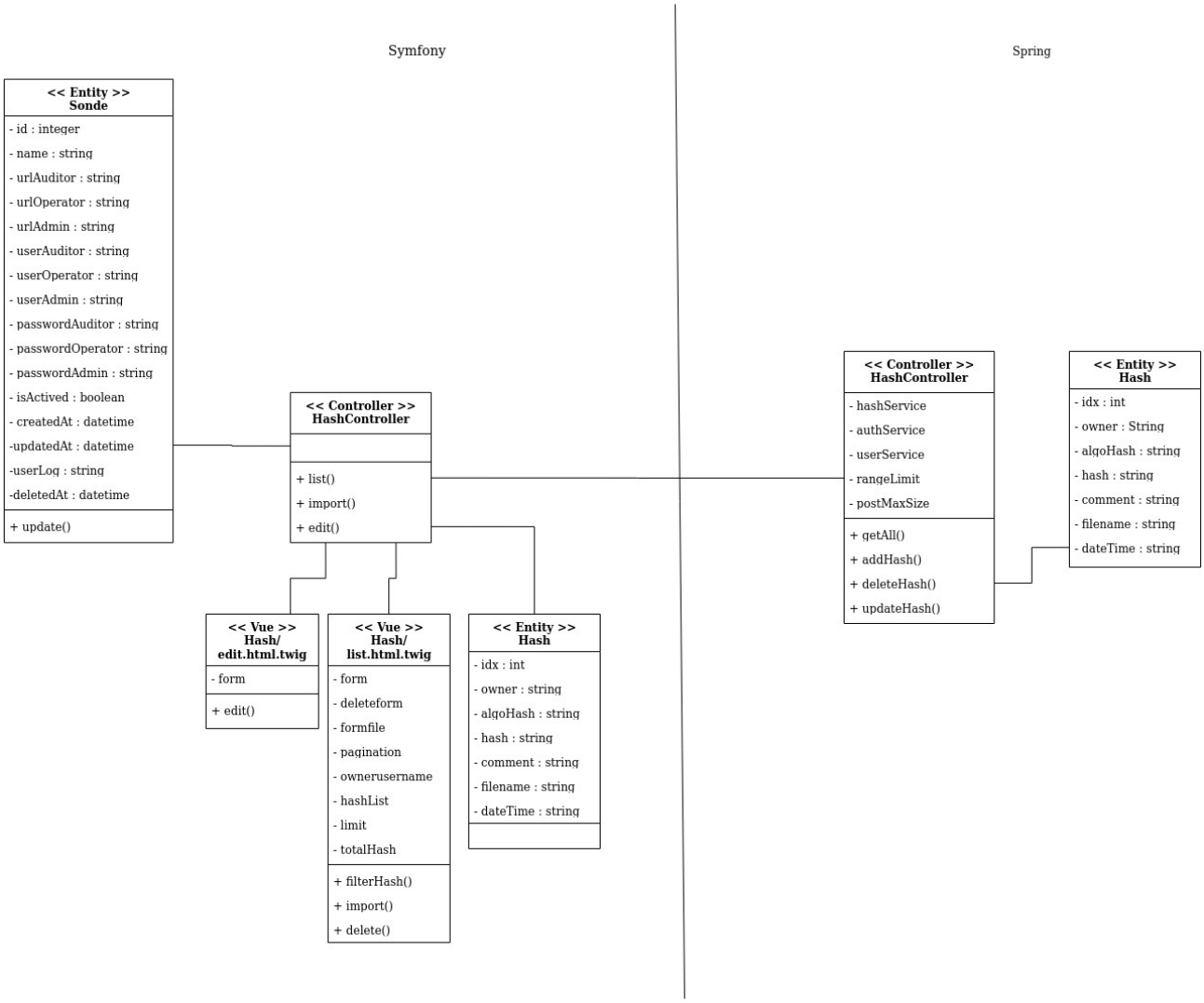
Diagramme de classes d'analyse du domaine

Afin de mieux comprendre le fonctionnement de la sonde ainsi que les différentes relations entre chaque classe, j'ai réalisé un diagramme de classe du domaine en intégrant le diagramme précédent :



Partie 4 – ANALYSE TECHNIQUE

Afin de traduire concrètement les diagrammes d'analyse fonctionnelle, je commence par réaliser un diagramme de classes de conception en considérant les 2 applications, l'application PHP via Symfony et l'application JAVA via Spring :



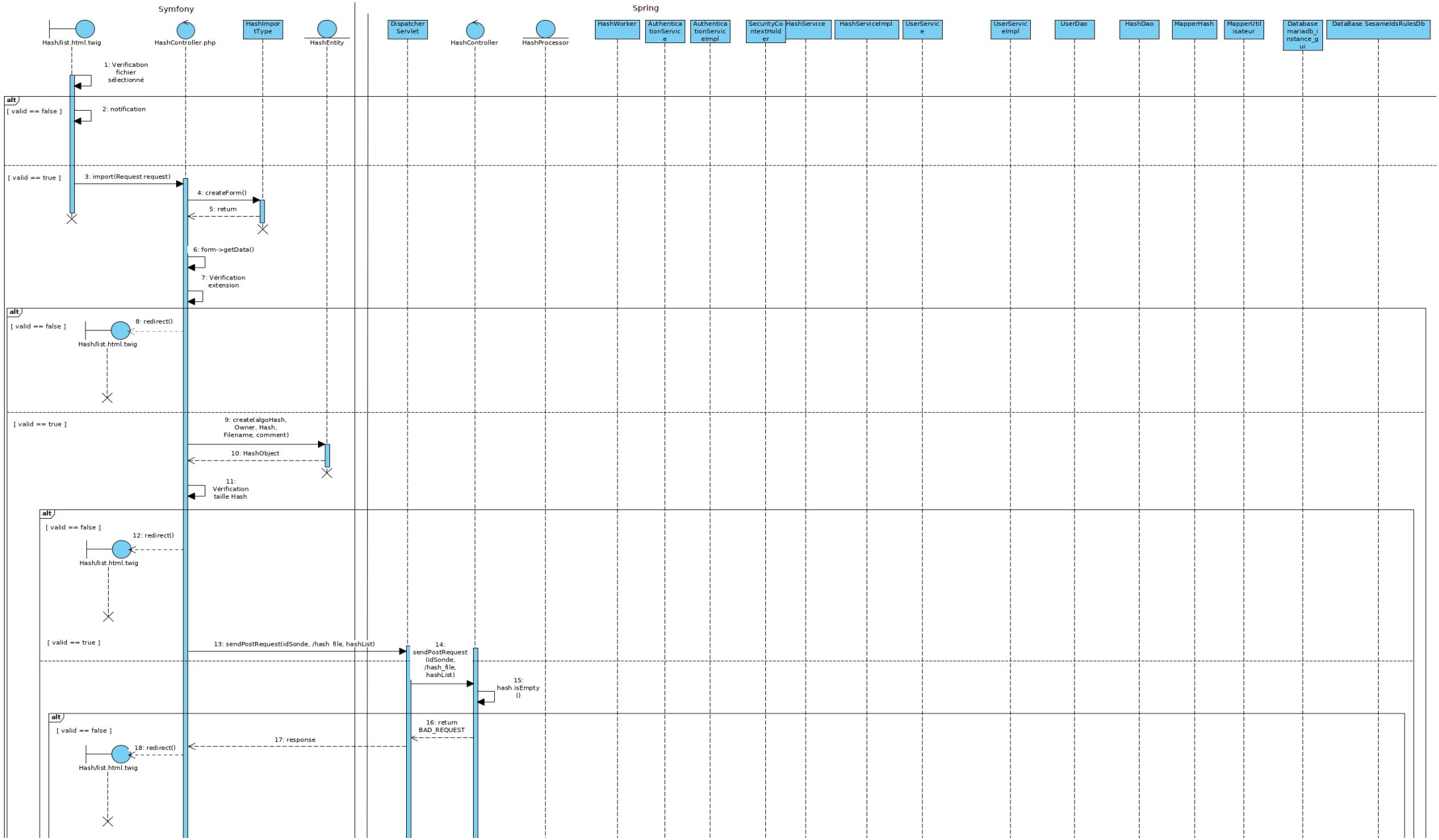
Analyse séquentielle

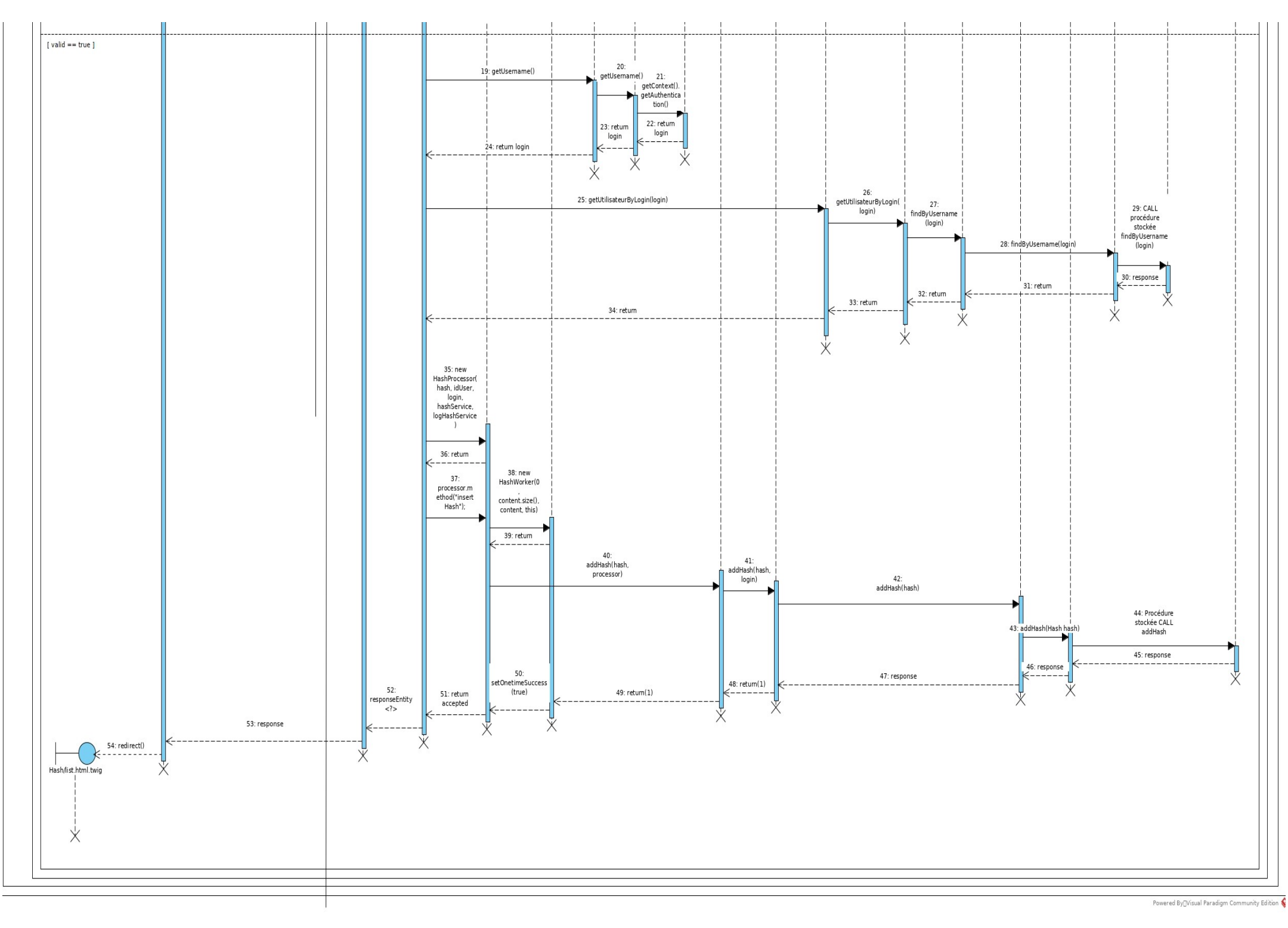
Dans l’objectif d’obtenir une analyse technique plus précise, j’ai fais des diagrammes de séquence pour le projet « Gérer Hash IDS/DPI ».

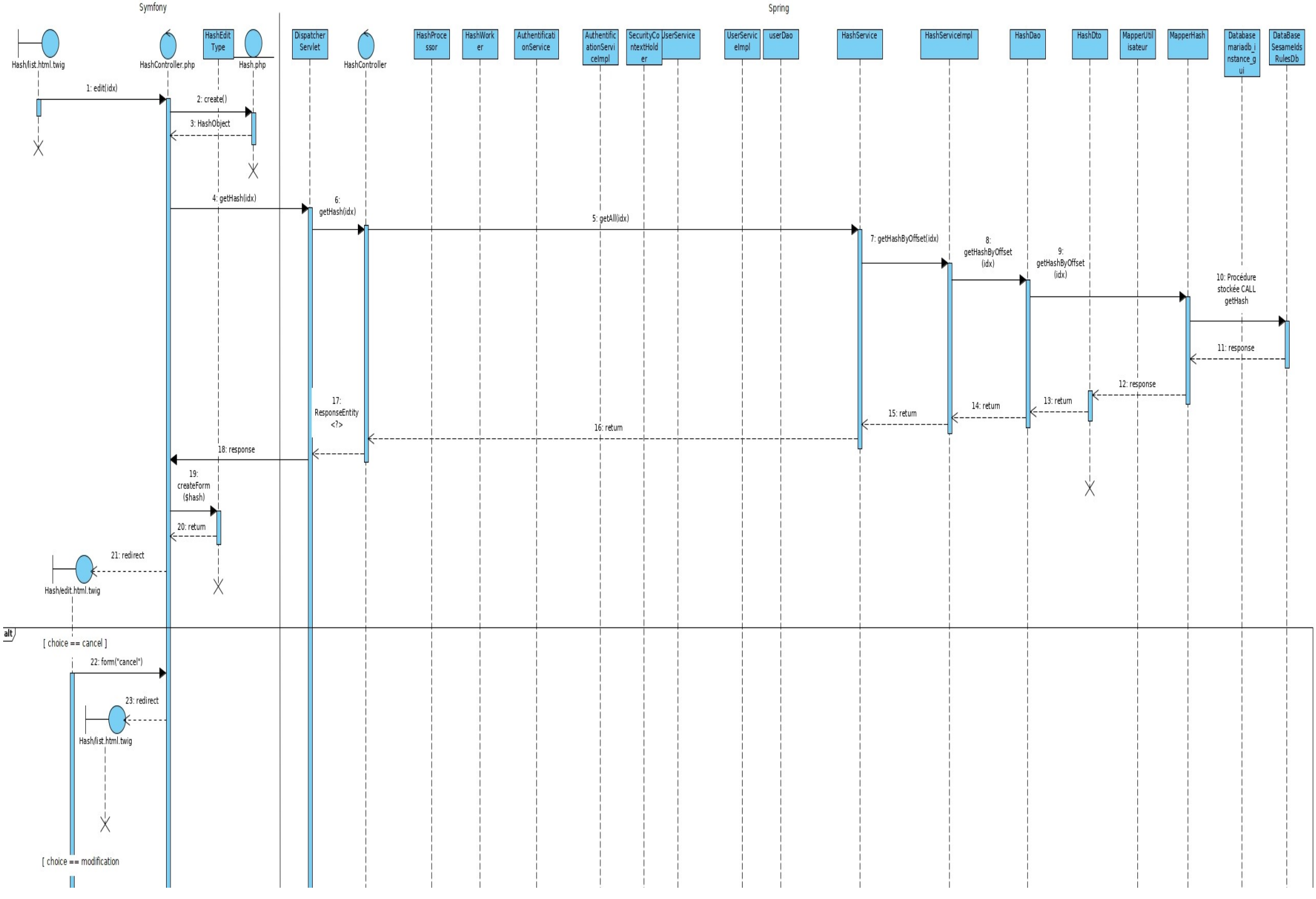
Chaque diagramme est présenté sous 2 pages pour plus de lisibilité.

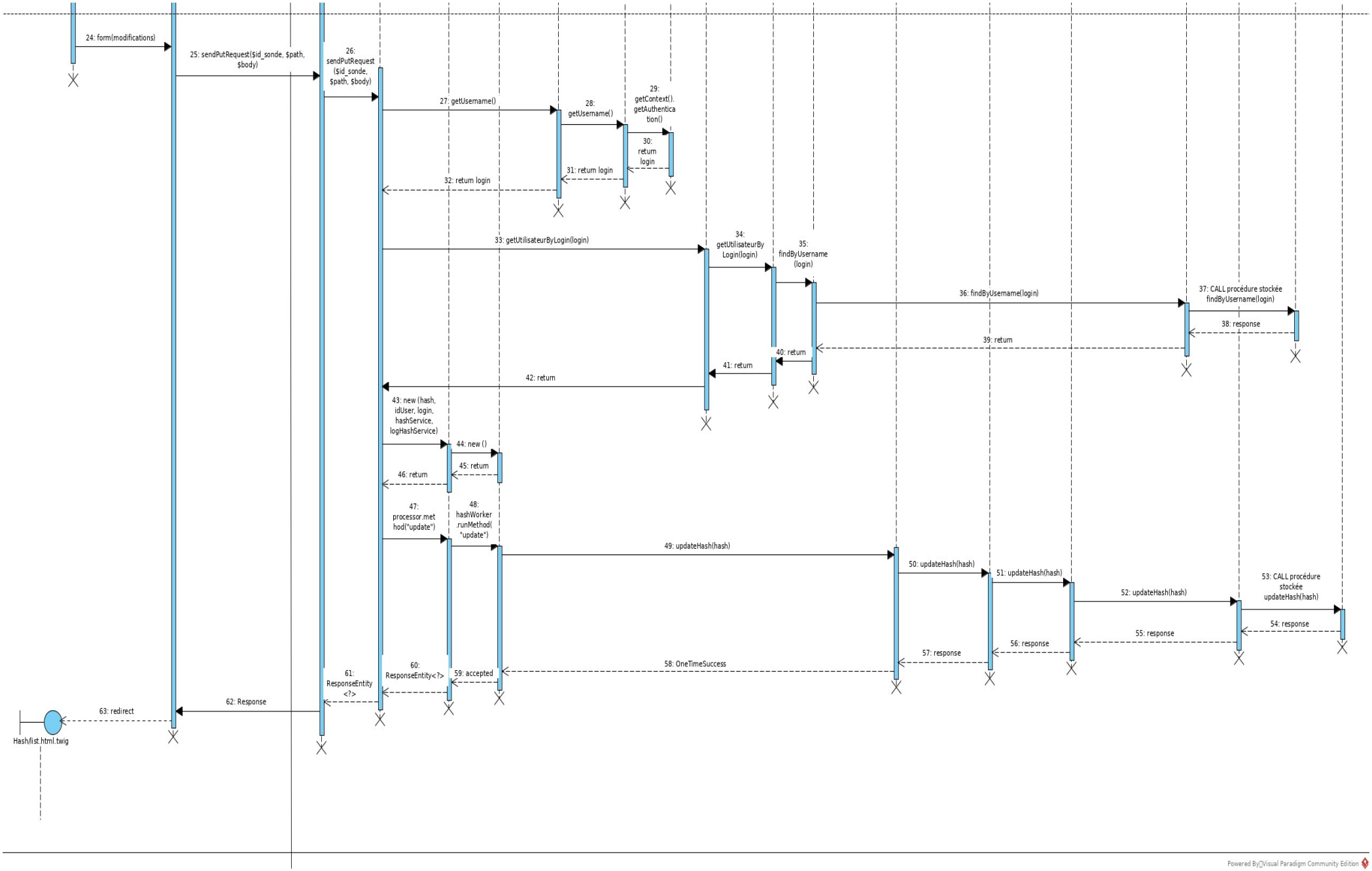
L’ordre chronologique des diagramme est :

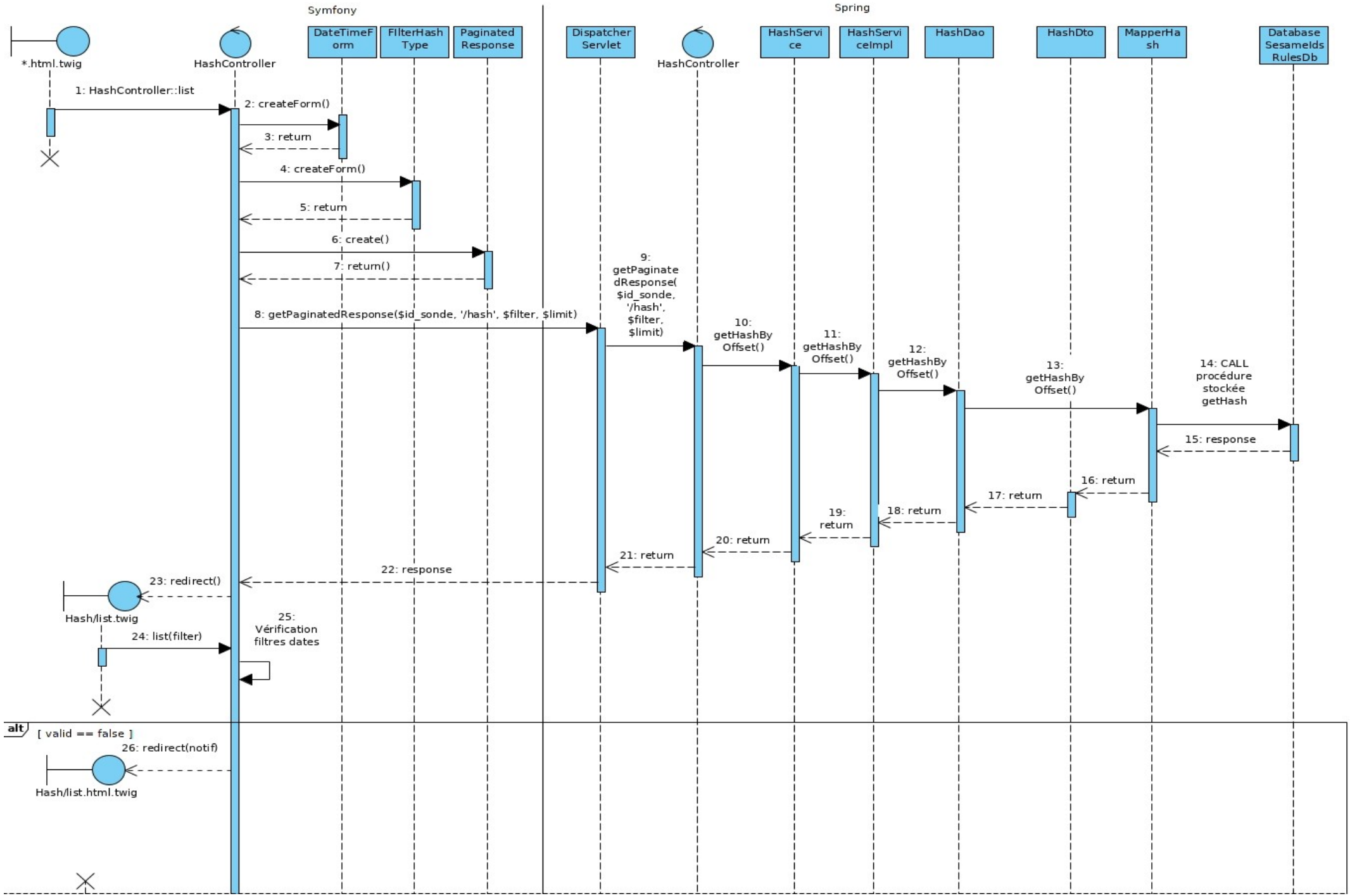
1. « Gérer Hash IDS/DPI » > ajouter hash.
2. « Gérer Hash IDS/DPI » > mettre à jour hash.
3. « Gérer Hash IDS/DPI » > afficher hash
4. « Gérer Hash IDS/DPI » > supprimer hash

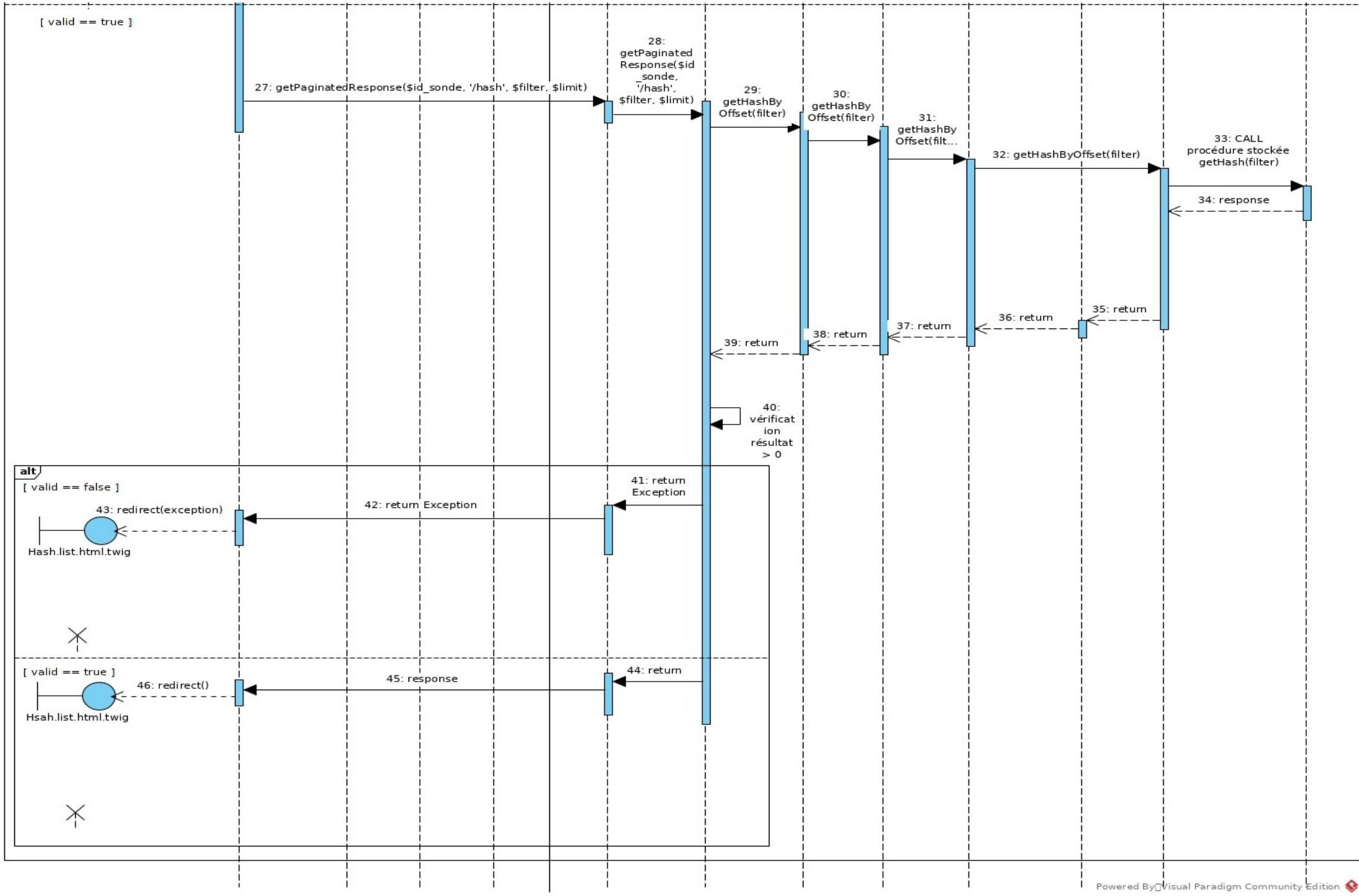

Symfony
Spring
Hash/list.html.twig
HashController.php
HashImportType
HashEntity
DispatcherServlet
HashController
HashProcessor
HashWorker
AuthenticationService
AuthenticationServiceImpl
SecurityContextHolder
HashService
HashServiceImpl
UserService
UserServiceImpl
UserDao
HashDao
MapperHash
MapperUtilisateur
Database mariadb_instance_gui
DataBase SesameIdsRulesDb
1: Verification fichier sélectionné
alt
[valid == false]
2: notification
[valid == true]
3: import(Request request)
4: createForm()
5: return
6: form->getData()
7: Vérification extension
alt
[valid == false]
8: redirect()
Hash/list.html.twig
[valid == true]
9: create(algoHash, Owner, Hash, Filename, comment)
10: HashObject
11: Vérification taille Hash
alt
[valid == false]
12: redirect()
Hash/list.html.twig
[valid == true]
13: sendPostRequest(idSonde, /hash_file, hashList)
14: sendPostRequest (idSonde, /hash_file, hashList)
15: hash.isEmpty ()
alt
[valid == false]
16: return BAD_REQUEST
17: response
18: redirect()
Hash/list.html.twig

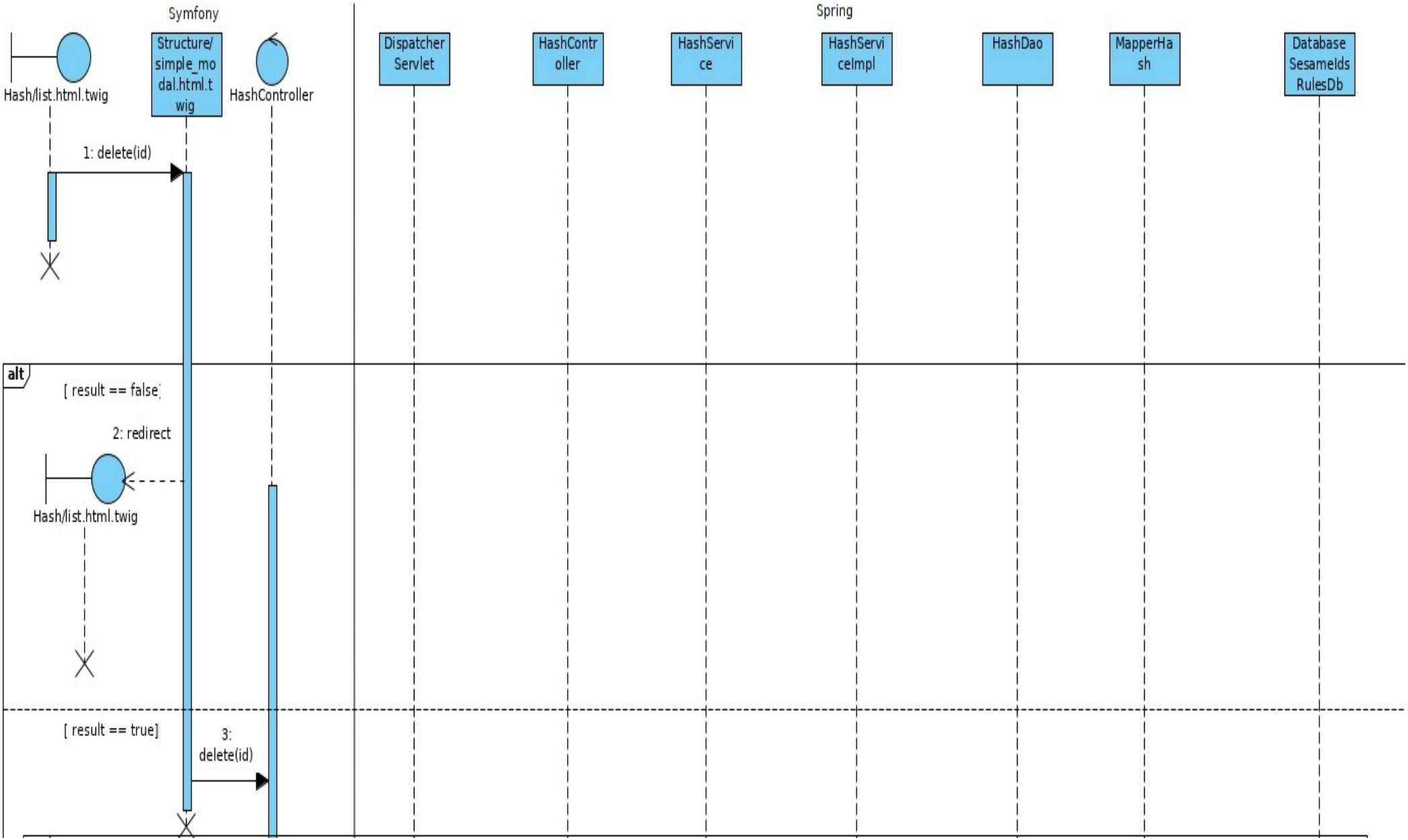


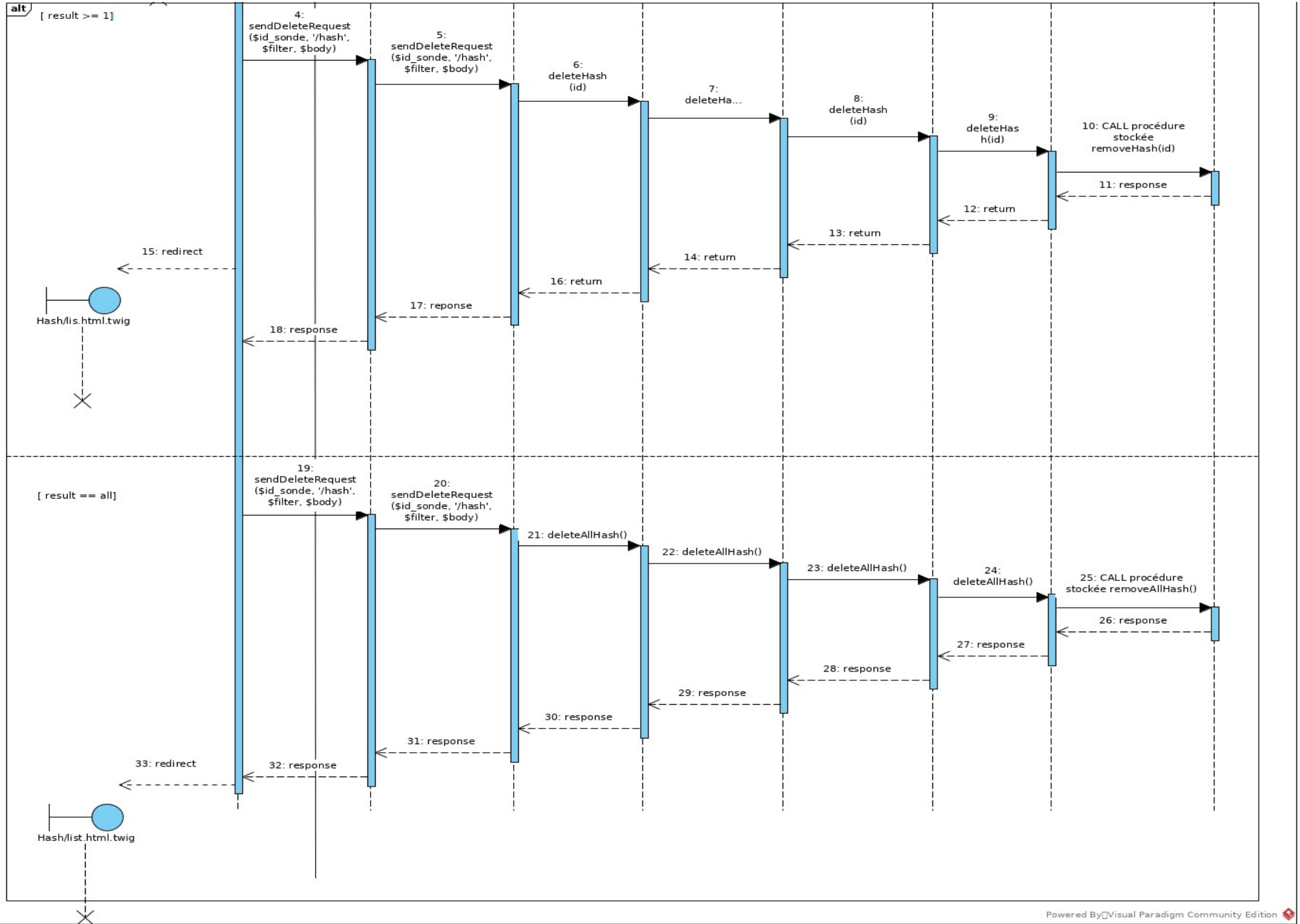

Symfony
Spring
Hash/list.html.twig
HashController.php
HashEdit Type
Hash.php
DispatcherServlet
HashController
HashProcessor
HashWorker
AuthentificationService
AuthentificationServiceImpl
SecurityContextHolder
UserService
UserServiceImpl
userDao
HashService
HashServiceImpl
HashDao
HashDto
MapperUtilisateur
MapperHash
Database mariadb_instance_gui
DataBase SesameIdsRulesDb
1: edit(idx)
2: create()
3: HashObject
4: getHash(idx)
6: getHash(idx)
5: getAll(idx)
7: getHashByOffset(idx)
8: getHashByOffset (idx)
9: getHashByOffset (idx)
10: Procédure stockée CALL getHash
11: response
12: response
13: return
14: return
15: return
16: return
17: ResponseEntity <?>
18: response
19: createForm ($hash)
20: return
21: redirect
Hash/edit.html.twig
alt
[choice == cancel]
22: form(“cancel”)
23: redirect
Hash/list.html.twig
[choice == modification




Symfony
Spring
*.html.twig
HashController
DateTimeForm
FIlterHashType
PaginatedResponse
DispatcherServlet
HashController
HashService
HashServiceImpl
HashDao
HashDto
MapperHash
Database SesameIds RulesDb
1: HashController::list
2: createForm()
3: return
4: createForm()
5: return
6: create()
7: return()
8: getPaginatedResponse($id_sonde, '/hash', $filter, $limit)
9: getPaginatedResponse($id_sonde, '/hash', $filter, $limit)
10: getHashBy Offset()
11: getHashBy Offset()
12: getHashBy Offset()
13: getHashBy Offset()
14: CALL procédure stockée getHash
15: response
16: return
17: return
18: return
19: return
20: return
21: return
22: response
23: redirect()
Hash/list.twig
24: list(filter)
25: Vérification filtres dates
alt
[valid == false]
26: redirect(notif)
Hash/list.html.twig


[valid == true]
27: getPaginatedResponse($id_sonde, '/hash', $filter, $limit)
28: getPaginated Response($id _sonde, '/hash', $filter, $limit)
29: getHashBy Offset(filter)
30: getHashBy Offset(filter)
31: getHashBy Offset(filt...
32: getHashByOffset(filter)
33: CALL procédure stockée getHash(filter)
34: response
35: return
36: return
37: return
38: return
39: return
40: vérificat ion résultat > 0
alt
[valid == false]
41: return Exception
42: return Exception
43: redirect(exception)
Hash.list.html.twig
[valid == true]
44: return
45: response
46: redirect()
Hsah.list.html.twig
Powered By Visual Paradigm Community Edition


Symfony
Spring
Hash/list.html.twig
Structure/ simple_mo dal.html.t wig
HashController
Dispatcher Servlet
HashContr oller
HashServi ce
HashServi ceImpl
HashDao
MapperHa sh
Database SesameIds RulesDb
1: delete(id)
alt
[result == false]
2: redirect
Hash/list.html.twig
[result == true]
3: delete(id)



La base de données

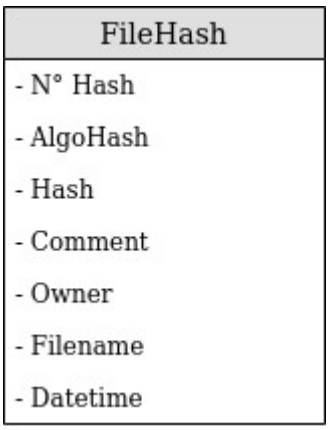
Concernant l'application qui gère la sonde de détection, plusieurs bases de données sont utilisées pour les différentes fonctions. Pour le projet « Gérer Hash IDS/DPI », j'ai dû utiliser la base « SesameIdsRulesDb » où plusieurs tables sont déjà présentes.

Cependant, il n'existe aucune relation entre chaque table car chaque entité est indépendante. Cette structure est justifiée par plusieurs mesures de sécurité pour répondre aux exigences de la certification.

Les communications entre la base de données et l'application JAVA se font via des procédures stockées. Ces procédures stockées sont plus sécurisées pour les attaques de type « injection SQL ». Elles sont aussi plus optimisées permettant de diminuer les aller-retour entre le client et le serveur.

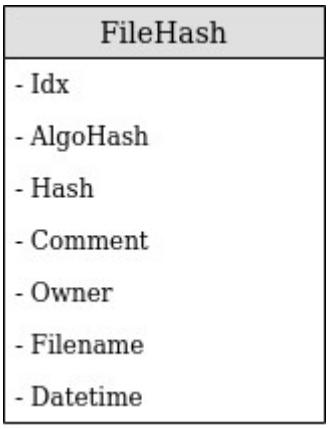
Pour concevoir et réaliser la table dans la base de données, j'ai réalisé plusieurs diagramme.

Modèle Conceptuel de données (MCD)

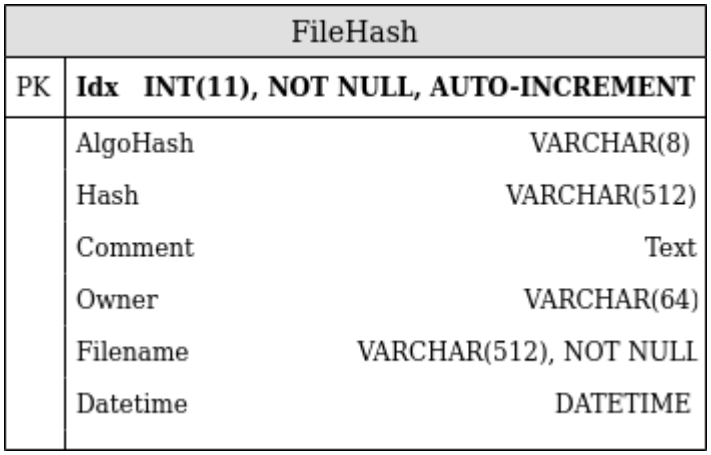


Modèle Logique de données (MLD)

Le MLD est élaboré à partir du MCD, qui permet de transformer chaque entité en table afin que ça puisse être compréhensible par un SGBD (Système de Gestion de Base de Données).



Modèle physique de données (MPD)



Dictionnaire de données

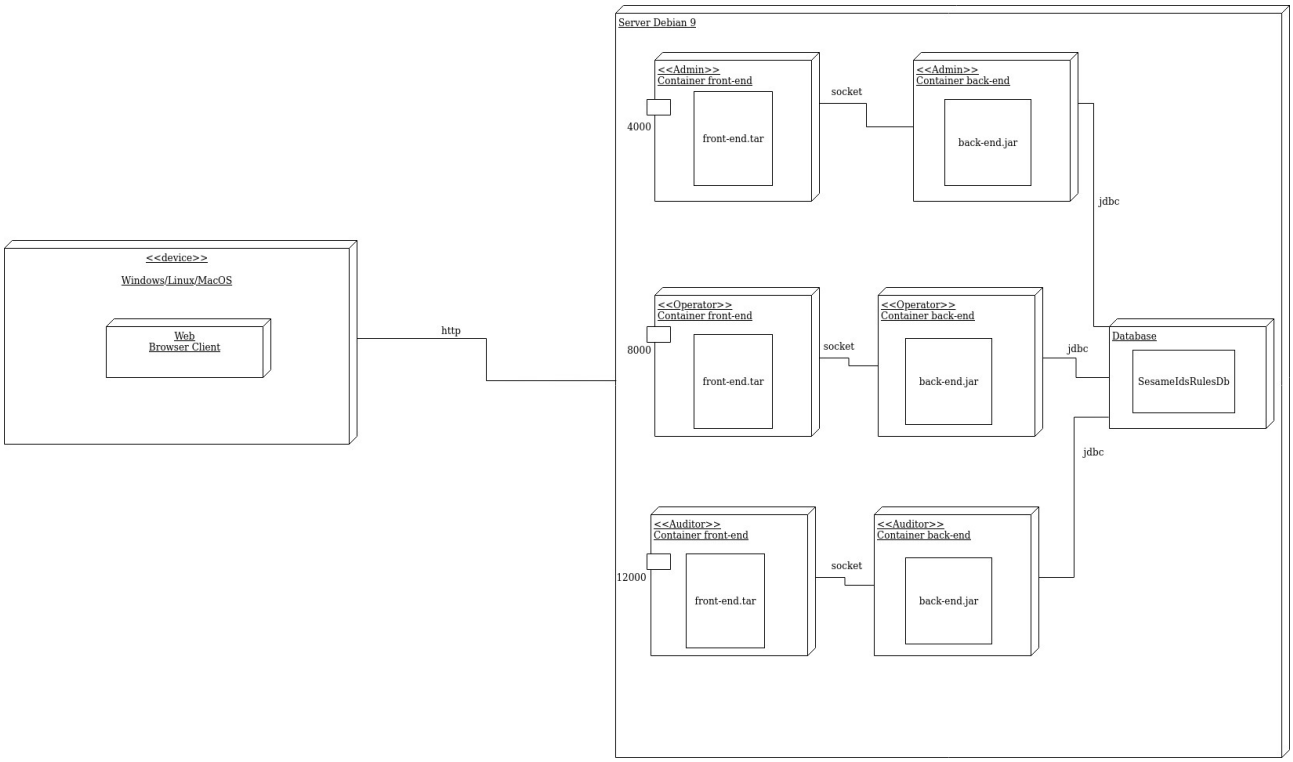
J'ai réalisé un dictionnaire de données de la base de données que j'ai utilisé, voici l'exemple de la table « FileHash » :

Colonne	Remarque	Type	Null / NotNull	Autres	Commentaire
Idx	Primary Key	INT	NOT NULL	AI	Identifiant du hash
AlgoHash		VARCHAR(8)	NOT NULL		Algorithme qui a chiffré le hash
Hash		VARCHAR(64)	NOT NULL		Hash
Comment		Text	NULL		Commentaire lors de l'ajout du hash
Owner		VARCHAR(11)	NOT NULL		L'acteur qui a ajouté le hash
Filename		VARCHAR(512)	NOT NULL		Le nom du fichier qui a ajouté le hash
Datetime		DATETIME	NOT NULL		Date à laquelle le hash a été ajouté

Architecture

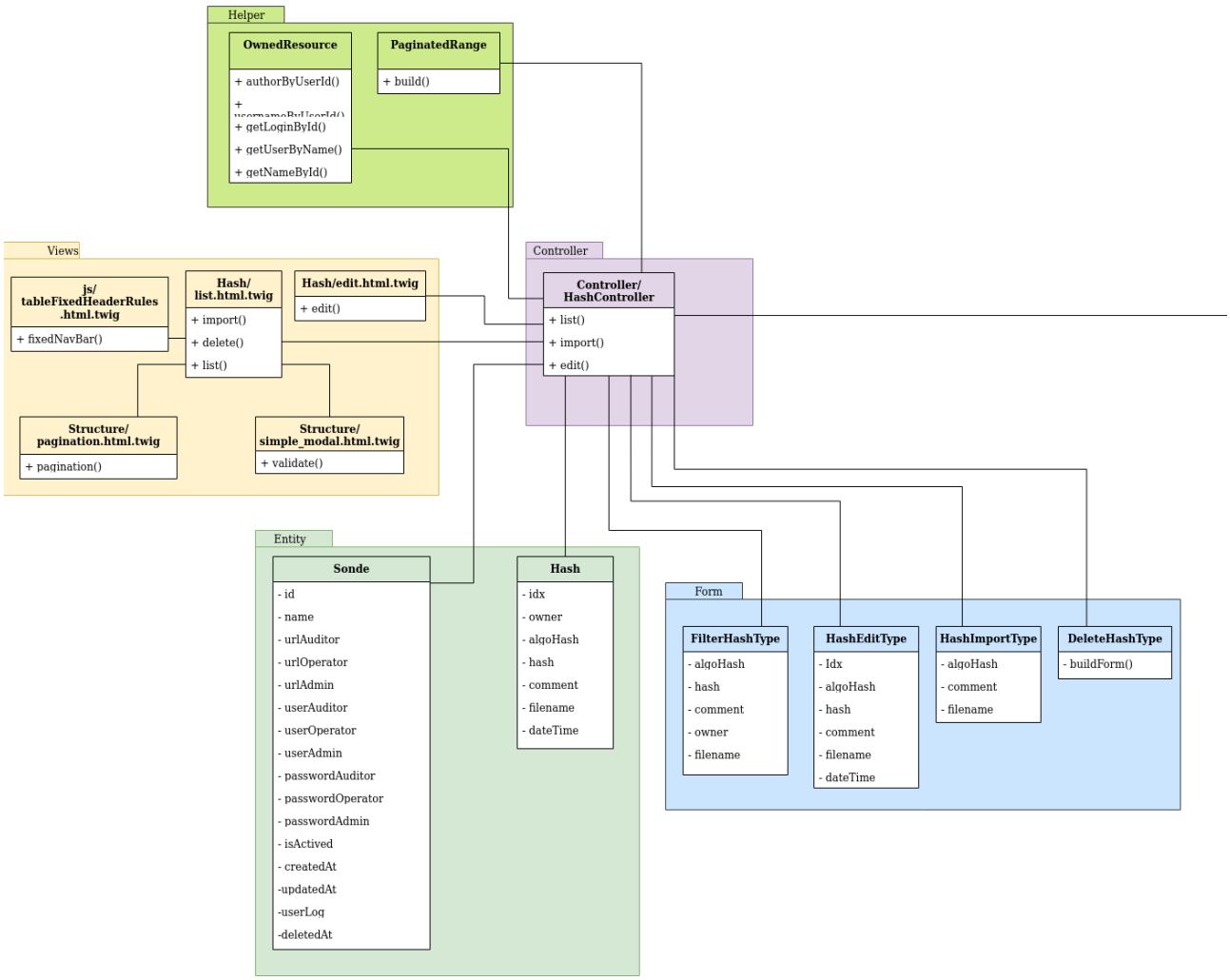
La sonde est basée sur l'architecture Microservices. Cette architecture permet découper une application en petits services, appelés Microservices, parfaitement autonomes qui exposent une API que les autres Microservices pourront consommer.

La particularité de la sonde est qu'elle est centralisée. L'application PHP et JAVA sont mises séparément dans des containers LXC respectifs, ainsi que les différentes bases de données dont celle que j'ai utilisé soit « SesameIdsRulesDb ».

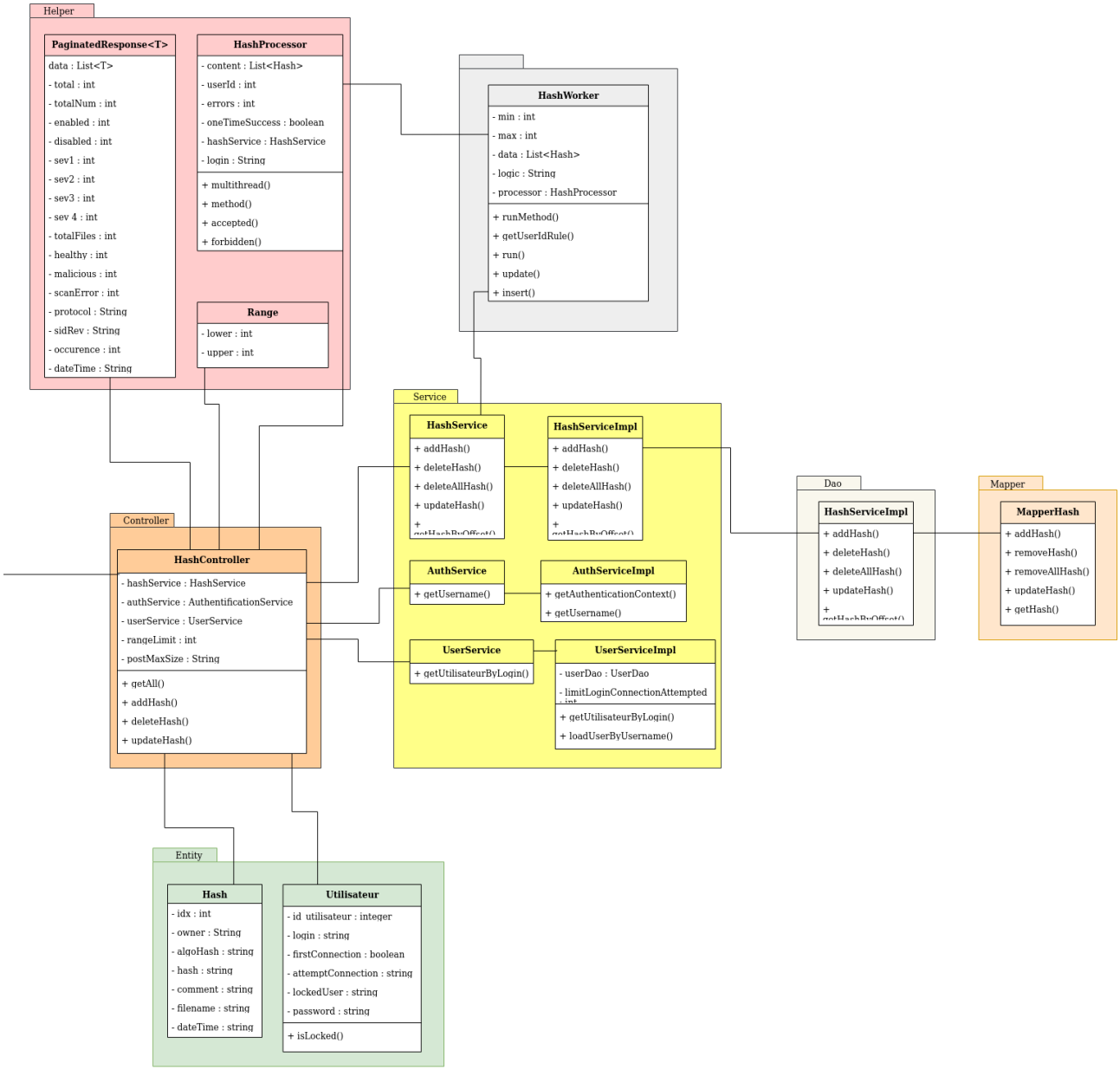


Packages

Pour commencer le développement, j'ai dû respecter l'arborescence déjà mise en place. Pour Symfony :



Pour Spring :


Helper
PaginatedResponse<T>
data : List<T>
- total : int
- totalNum : int
- enabled : int
- disabled : int
- sev1 : int
- sev2 : int
- sev3 : int
- sev 4 : int
- totalFiles : int
- healthy : int
- malicious : int
- scanError : int
- protocol : String
- sidRev : String
- occurence : int
- dateTime : String
HashProcessor
- content : List<Hash>
- userId : int
- errors : int
- oneTimeSuccess : boolean
- hashService : HashService
- login : String
+ multithread()
+ method()
+ accepted()
+ forbidden()
Range
- lower : int
- upper : int
HashWorker
- min : int
- max : int
- data : List<Hash>
- logic : String
- processor : HashProcessor
+ runMethod()
+ getUserIdRule()
+ run()
+ update()
+ insert()
Service
HashService
+ addHash()
+ deleteHash()
+ deleteAllHash()
+ updateHash()
HashServiceImpl
+ addHash()
+ deleteHash()
+ deleteAllHash()
+ updateHash()
Dao
HashServiceImpl
+ addHash()
+ deleteHash()
+ deleteAllHash()
+ updateHash()
Mapper
MapperHash
+ addHash()
+ removeHash()
+ removeAllHash()
+ updateHash()
+ getHash()
Controller
HashController
- hashService : HashService
- authService : AuthentificationService
- userService : UserService
- rangeLimit : int
- postMaxSize : String
+ getAll()
+ addHash()
+ deleteHash()
+ updateHash()
AuthService
+ getUsername()
AuthServiceImpl
+ getAuthenticationContext()
+ getUsername()
UserService
+ getUtilisateurByLogin()
UserServiceImpl
- userDao : UserDao
+ getUtilisateurByLogin()
+ loadUserByUsername()
Entity
Hash
- idx : int
- owner : String
- algoHash : string
- hash : string
- comment : string
- filename : string
- dateTime : string
Utilisateur
- id_utilisateur : integer
- login : string
- firstConnection : boolean
- attemptConnection : string
- lockedUser : string
- password : string
+ isLocked()

Développement

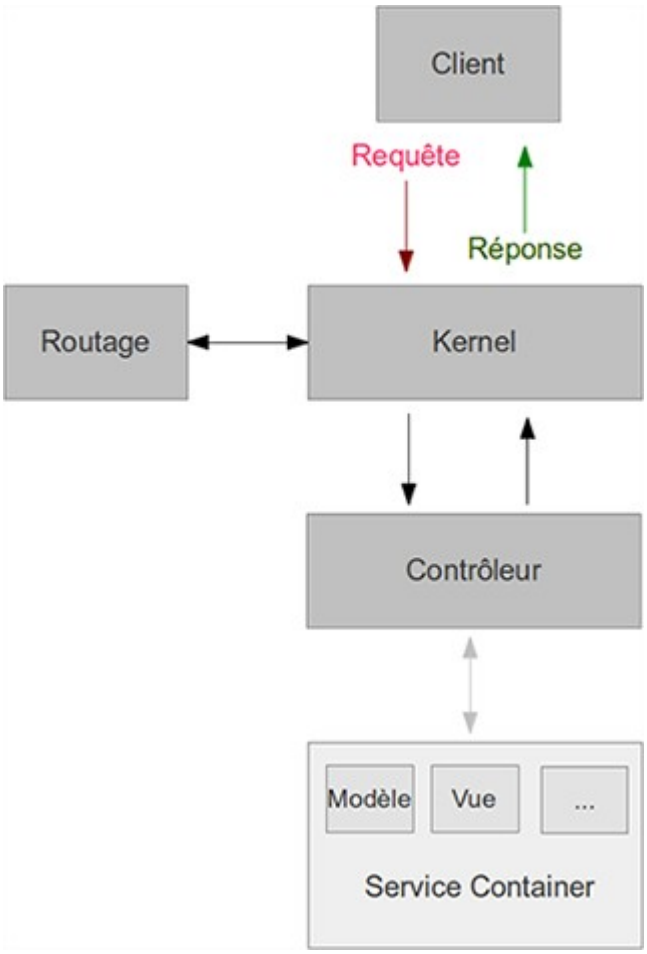
Concernant le développement, j'ai été dans l'obligation logique de respecter les différents langages déjà mis en place soit :

- PHP 7.1
- JAVA 8

Les différents langages sont utilisés via les frameworks suivants :

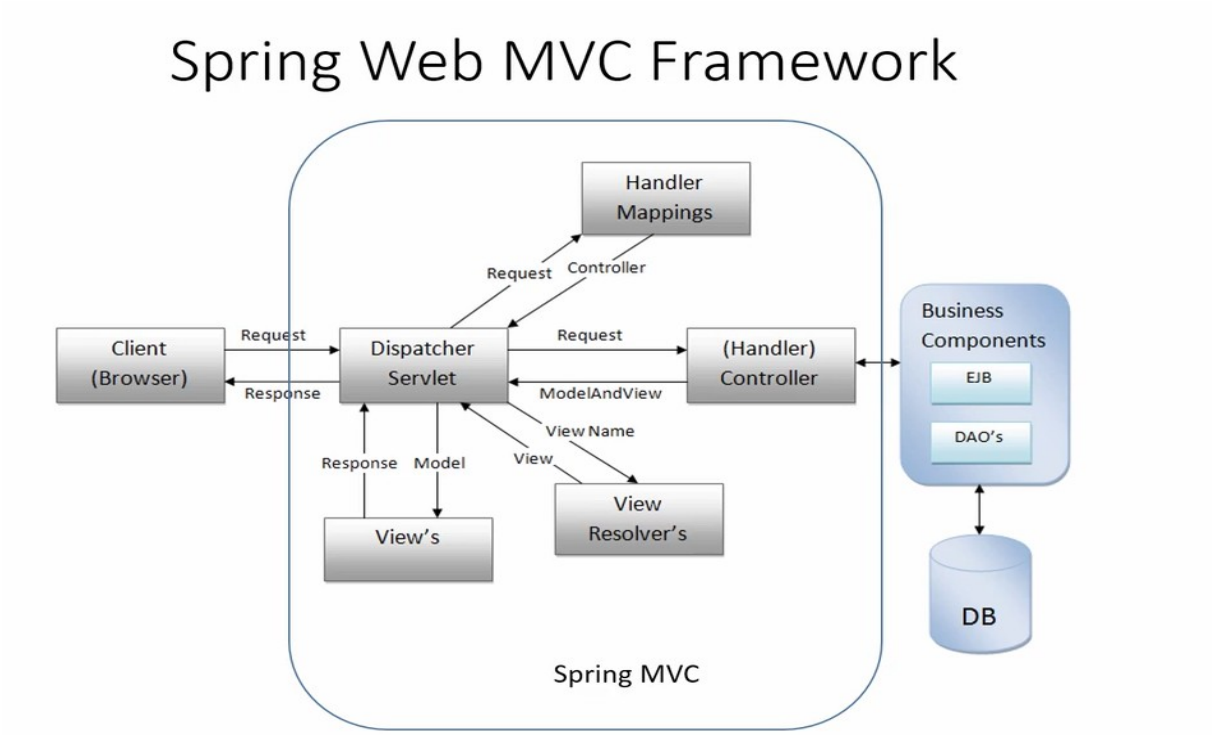
- Symfony 4.2.3

  Symfony est un ensemble de composants PHP fonctionnant entre-eux, fonctionnant sur une architecture MVC. Symfony 4 est optimisé pour l'installation via composer pour un déploiement plus rapide et simple.



- Spring 2.0.5

  Spring est un framework utilisant le langage JAVA basé sur l'API servlet de JAVA JEE permettant de simplifier le développement d'applications web en respectant le patron de conception MVC 2.



Pour coder, j'ai utilisé les IDE déjà mis en place soit Visual Studio Code, et Eclipse.

L'outil de gestion de version utilisé est GIT 2.11.0. A savoir que le GIT est local dans un serveur de chez SesameIT.

Les bases de données sont gérés par MariaDb 10.1.41, et la gestion des bases se fait via le terminal.

Pour réaliser le développement et le codage, je me suis inspiré des autres classes déjà codées, comme par exemple « Rules IDS/DPI » ou « YaraRule IDS/DPI », ce qui m'a permis d'intégrer mon code avec plus d'homogénéité.

Exemple de réalisation n°1 : Ajouter un hash

Dans un premier temps, je commence par l'implémentation dans l'application PHP, pour générer la vue, le traitement des informations saisies par l'utilisateur, etc.

Je commence par coder la classe Hash.php (les accesseurs et mutateurs ont été volontairement réduits afin d'alléger les fichiers):

```php
<?php

namespace App\Helper;


use App\Exception\UserIdNotProvidedException;
use App\Helper\Suricata\Parser;
use JsonSerializable;

class Hash implements JsonSerializable {
    private $idx;
    private $owner;
    private $algoHash;
    private $hash;
    private $comment;
    private $filename;
    private $dateTime;

    public static function create() : Hash {
        return new Hash();
    }

    public function getIdx()
    {
        return $this->idx;
    }

    public function setIdx($idx)
    {
        $this->idx = $idx;
        return $this;
    }

    public function getOwner()
    {…
    }

    public function setOwner($owner)
    {…
    }
```

```php
    public function jsonSerialize()
    {
        $toJson = [
            'idx' => $this->idx,
            'owner' => $this->owner,
            'algoHash' => $this->algoHash,
            'hash' => $this->hash,
            'comment' => $this->comment,
            'filename' => $this->filename,
            'dateTime' => $this->dateTime,

        ];

        return $toJson;
    }
}
```

Après avoir créé la classe Hash.php, je dois coder le controlleur HashController.php. Pour créer le controlleur HashController.php, il est nécessaire de connecter le controlleur avec l'URI dans le fichier account.yml :

```yaml
sonde_hash_import:
    path: /sonde/{id_sonde}/import/hash
    controller: App\Controller\HashController::import
    requirements:
        id_sonde: \d+
    methods: [POST]
```

Je peux donc commencer à coder le controlleur et pour que l'import d'un hash puisse être possible il est nécessaire de créer un formulaire.

La création du formulaire via Symfony permet de se protéger des attaques de type CSRF grâce à l'ajout d'un champs caché nommé « csrf_token » qui contient un jeton de validation de formulaire généré par le serveur. De plus, ça permet de gérer la validation des données.

Je fais donc appel au builder pour créer mon formulaire nommé HashImportType.php pour respecter les règles de nommages déjà mises en place :

```php
<?php

namespace App\Form\Type;


use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\FileType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;


class HashImportType extends AbstractType
{

    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('file', FileType::class, [
                'label' => ' ',
                'attr' => [
                    'class' => 'mb-3',
                    'for' => 'customFile'
                ]
            ])
            ->add('comment', TextType::class, [
                'required'=>false,
                'attr'=>[
                    'class'=>'form-control form-control-sm',
                    'maxlength' => 150
                ]
            ])
            ->add('importAlgo', ChoiceType::class, [
                'required'=>true,
                'label'=>'AlgoHash',
                'attr'=>[
                    'class'=>'form-control form-control-sm'
                ],
                'choices' => [
                    'MD5' => 'MD5',
                    'SHA1' => 'SHA1',
                    'SHA256' => 'SHA256',
                ],
            ])
            ->add('save', SubmitType::class, [
                'label'    => 'Upload',
                'attr'     => [
                    'class'=>'btn btn-sm btn-warning mb-3'
                ]
            ]);
    }
}

```

La vue est générée par le moteur de templates TWIG, qui permet de se protéger contre l'attaque de type XSS ainsi que d'éviter de se faire voler la session d'utilisateur. Symfony créé le cookie qui contient l'id de session avec l'option « http_only » qui prend la valeur « true ». De plus, TWIG échappe les caractères spéciaux de HTML.

La page Hash/list.twig.html doit être accessible par un « dropdown » dans la barre de navigation. Pour cela, j'ajoute l'« item » qui va permettre de rediriger sur la page Hash/list.html.twig. Cet ajout se fait sur la vue menu.html.twig.

La fonctionnalité d'import d'un hash doit être présent sur la même page que la liste des hashs. Il faut donc intégrer le formulaire à la page Hash/list.html.twig. Cette vue est la première vue qui a été codé sur le projet.

```
<div>
    <div class="col-6">
    <p>
        <b>Hash Upload</b>
    </p>
    </div>
    {{ form_start(form_file) }}
    <div class="col">
    {{ form_label(form_file.file) }}
    {{ form_widget(form_file.file) }}
    </div>
    <div class="col-sm-10">
    {{ form_label(form_file.importAlgo) }}{{ form_widget(form_file.importAlgo) }}
    </div>
    <div class="col-sm-10">
    {{ form_label(form_file.comment) }}
    <textarea maxlength="150" class="form-control form-control-sm"></textarea>
    <p>150 characters maximum</p>
    </div>

    <div class="col">
    {{ form_label(form_file.save) }}
    {{ form_widget(form_file.save) }}
    </div>
    {% do form_file.comment.setRendered %}
    {{ form_end(form_file) }}
</div>
```

L'application PHP est donc fonctionnelle, mais le back-end n'est pas encore implémenté et donc ne peut pas traiter les informations qui lui sont envoyées.

Je continue en implémentant l'application JAVA qui va traiter les informations envoyées par l'application PHP, et ensuite communiquer avec la base de données si nécessaire.

Dans un premier temps, je créé la classe Hash.java :

```java
package com.sesameit.entity;

public class Hash {
    private int idx;
    private String owner;
    private String algoHash;
    private String hash;
    private String comment;
    private String filename;
    private String dateTime;

    public Hash() {
    }

    public int getIdx() {
        return idx;
    }

    public void setIdx(int idx) {
        this.idx = idx;
    }
```

Afin d'alléger le contenu, j'ai réduit les accesseurs et mutateurs de la classe. De plus, une méthode ToString() a été implémentée.

Après avoir implémenté la méthode add() de la classe HashController.java, j'ai codé l'interface HashService.java et la classe HashServiceImpl.java qui vont faire la relation entre le controlleur et la classe de type service Dao (Data Access Object) nommée HashDao.java.

Cette classe HashDao.java implémente la méthode addHash() avec le hash en question passé en paramètre. Elle fait la relation avec le fichier XML MapperHash.xml.

Dans le fichier XML MapperHash.xml qui permet de communiquer avec la base de données, une méthode addHash() est implémentée, afin d'appeler la procédure stockée de la base de données nommée aussi addHash() :

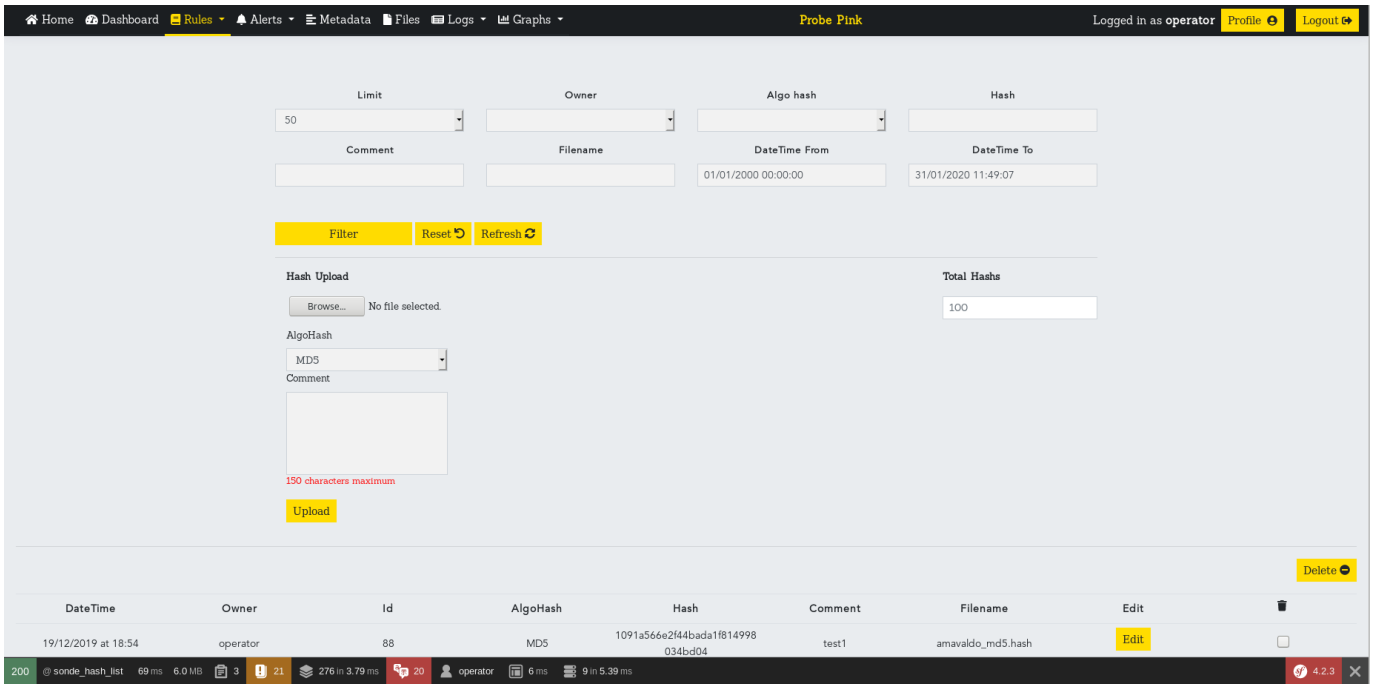
```xml
    <insert id="addHash" parameterType="com.sesameit.entity.Hash" keyColumn="idx">
        CALL addHash(#{algo_hash}, #{hash}, #{owner}, #{filename}, #{comment});
    </insert>
```

Après avoir implémenté toutes les classes, services, et fichiers XML nécessaires pour que les 2 applications puissent communiquer, je dois écrire la procédure stockée qui va permettre de créer la relation entre l'application JAVA et la base de données :

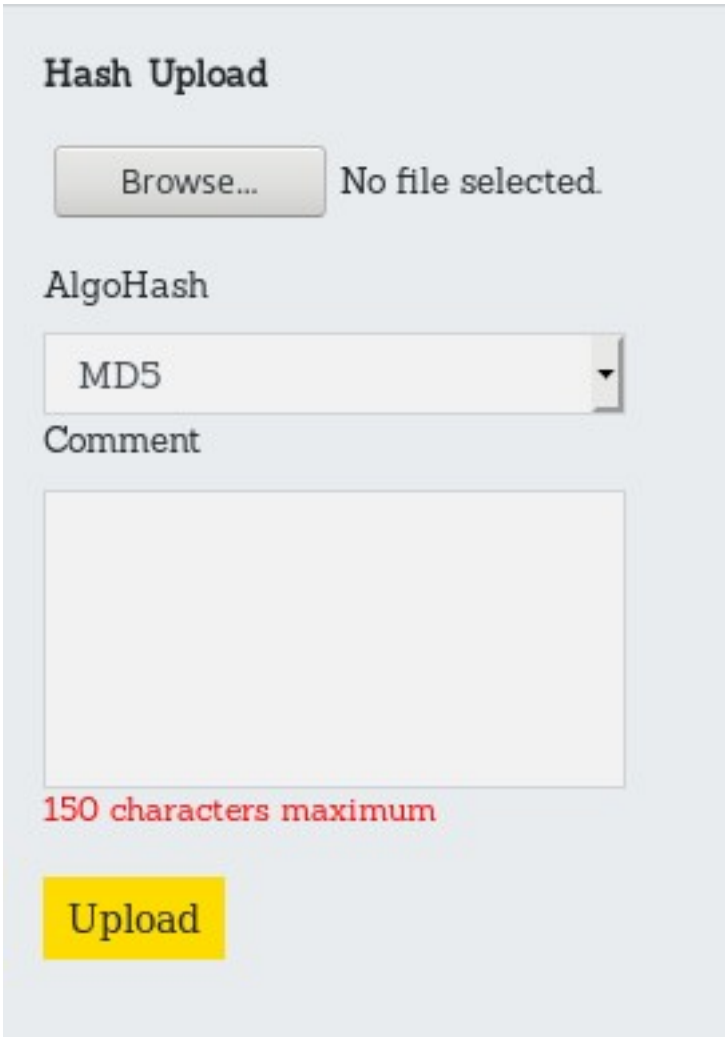
```
CREATE PROCEDURE addHash (IN p_algo VARCHAR(8),
                          IN p_hash VARCHAR(64),
                          IN p_owner VARCHAR(11),
                          IN p_filename VARCHAR(512),
                          IN p_comment TEXT)
BEGIN
    INSERT INTO FileHash VALUES (NULL, p_algo, p_hash, p_comment, p_owner, p_filename, NOW());
END
```

Pour des raisons de sécurité, je n'ai pas pu montrer l'intégralité du code souhaité.

Voici le résultat de l'intégration du formulaire dans la page Hash/list.twig.html. Le résultat correspond majoritairement à la maquette réalisée grâce à la précision des informations qui m'ont été données (charte graphique, structure de page,etc) :



Voici un focus du formulaire fonctionnel :



Exemple de réalisation n°2 : Supprimer un hash

La classe HashController.php étant déjà partiellement codé, je peux passer à l'étape suivante.

Comme pour l'ajout d'un hash, il est nécessaire de déclarer dans le fichier account.yml, la méthode à exécuter si l'utilisateur entre les informations nécessaires dans l'URI (en l'occurrence, cela se fait via un bouton).

```
227   sonde_hash_edit:
228       path: /sonde/{id_sonde}/hash/{id}/edit
229       controller: App\Controller\HashController::edit
230       requirements:
231           id: \d+
232           id_sonde: \d+
```

Ensuite, je passe à l'implémentation de la méthode edit() du HashController.php.

La suppression passe via un formulaire nommé HashEditType.php. La suppression se fait à l'aide d'un checkbox associé à chaque hash et d'une validation. Voici le code implémenté dans la page Hash/list.html.twig :

```
<td class="text-center align-middle">
  <label for="hashToDeletes[{{ hash.idx }}]"></label>
  <input id="hashToDeletes[{{ hash.idx }}]" class="element-checkable" type="checkbox" name="hashToDeletes[]" value="{{ hash.idx }}">
</td>
```

Pour que la méthode edit() puisse savoir si l'utilisateur à choisi de supprimer un ou des hashs via le formulaire généré préalablement, je vérifie si le formulaire est envoyé et valable :

```
113                 if ($deleteForm->isSubmitted() && $deleteForm->isValid()) {
114                     if (($hashToDelete = $request->get('hashToDeletes')) != null) {
```

Si le formulaire est valide, alors les informations sont envoyées à l'application JAVA.

Je peux maintenant passer au codage de la classe HashController.java.

Dans le controlleur HashController.java, j'implémente la méthode pour déterminer si l'utilisateur a choisit soit, de supprimer un ou plusieurs hashs, soit l'intégralité des hashs. Si le résultat est différent de l'intégralité des hashs, alors le contrôleur appellera une méthode deleteHash() avec les hashs sélectionnés par l'utilisateur. En revanche, si l'utilisateur choisit de supprimer tous les hashs, alors le contrôleur appellera la méthode deleteAllHash().

La méthode deleteHash() appelle l'interface HashService.java qui appelle elle-même la classe HashServiceImpl.java. Dans cette classe de type Service, j'implémente la méthode deleteHash.java qui va permettre de supprimer le ou les hashs sélectionnés :

```
@Override
public void deleteHash(String idx, String login) throws IOException {
    this.hashDao.deleteHash(idx);
}
```

La méthode deleteAllHash() appelle l'interface HashService.java qui appelle elle-même la classe HashServiceImpl.java. Dans cette classe de type Service, j'implémente la méthode deleteAllHash.java qui va permettre de supprimer tous les hashs :

```
@Override
public void deleteAllHash(String login) throws IOException {
    this.hashDao.deleteAllHash();
}
```

En fonction de la méthode appelée par la classe HashServiceImpl, 2 méthodes sont implémentées dans l'interface HashDao.java :

```
@Mapper
public interface HashDao {
    void deleteHash(@Param("idx") String idx);
    void deleteAllHash();
```

L'interface HashDao avec les 2 méthodes de suppression, fait appel au fichier XML HashMapper.xml qui va permettre la relation avec la base de données.

Voici le résultat dans HashMapper.xml :

```
<delete id="deleteHash">
    CALL removeHash(#{idx});
</delete>

<delete id="deleteAllHash">
  CALL removeAllHash();
</delete>
```

Pour que l'application JAVA puisse communiquer avec la base de données, il reste juste à coder les 2 procédures stockées correspondant à chaque méthode appelée.

Procédure stockée removeHash() :

```
CREATE PROCEDURE removeHash(IN p_idx INTEGER)
BEGIN
    DELETE FROM FileHash WHERE Idx=p_Idx;
END
```

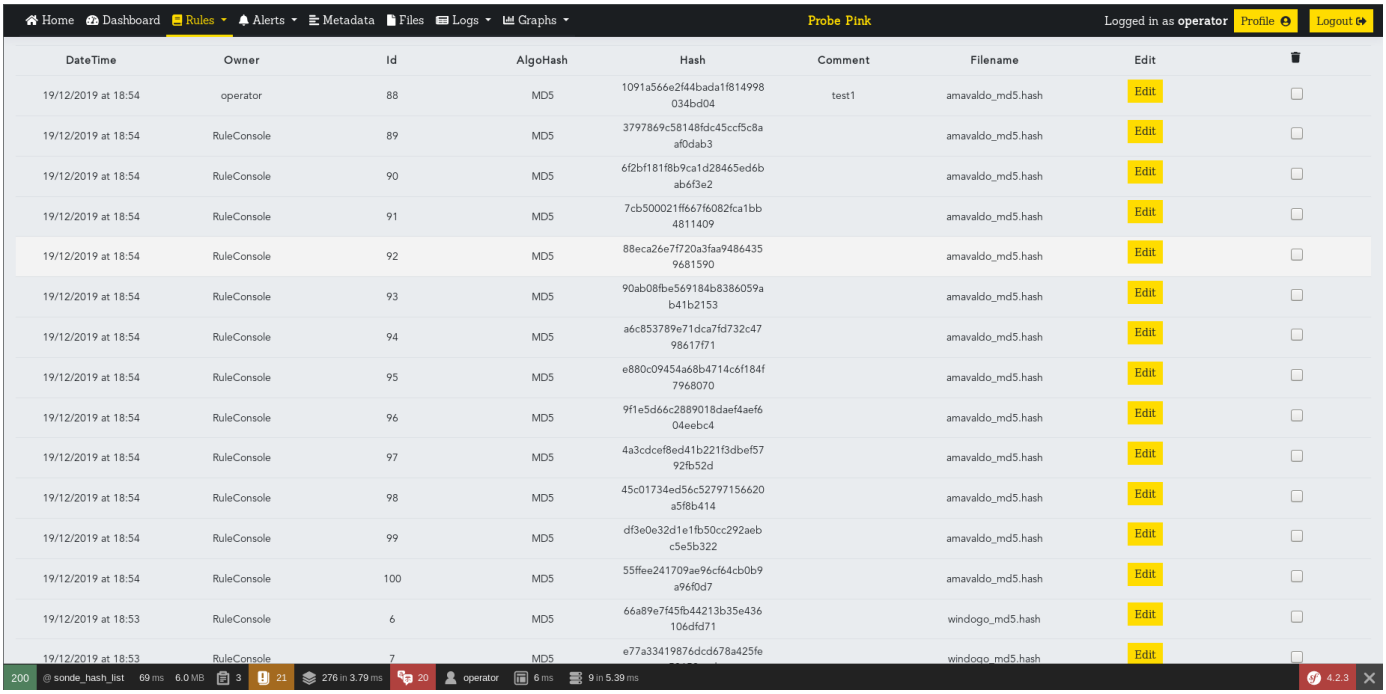
Procédure stockée removeAllHash() :

```
CREATE PROCEDURE removeAllHash()
BEGIN
    DELETE FROM FileHash;
END
```

Pour des raisons de sécurité, je n'ai pas pu montrer l'intégralité du code souhaité.

Le codage pour le cas d'utilisation « Gérer Hash IDS/DPI » > supprimer hash est à présent terminé, il reste à tester les 2 méthodes.

Voici l’IHM permettant de supprimer un ou plusieurs hashs :



Non présent sur la capture d’écran, un bouton permettant de déclencher la méthode est présent au dessus de la colonne des checkboxs. Un autre bouton est aussi présent permettant de sélectionner tous les hashs.

Les tests

En raison de l'évènement du Forum International de Cybersécurité, le projet a glissé de plusieurs jours. Par conséquence, je n'ai pas pu participer ni faire tous les tests nécessaires dans le développement d'une application ou d'un module d'une application.

Toutefois, pendant le développement, j'ai pu assurer les tests unitaires des différentes méthodes que j'ai implémentée. Pour ces tests, je n'ai utilisé aucun framework par manque de temps. J'ai entrée les données attendues par la méthode et j'ai comparé les données sortantes à celles théoriquement attendues. Lors de mon dernier jour, tous les tests unitaires du projet ont été validés.

De plus, au fur et à mesure du développement, j'ai effectué continuellement des tests de non-régression. Ils m'ont permis de savoir si le code que j'implémentais n'impactait pas le reste de l'application. Ces tests ont été aussi validés.

Après mon départ, j'ai reçu des premiers bons retours sur d'autres tests qui sont effectués par Maud soit les tests d'intégration et de performance.

CONCLUSION

Sur le respect du cahier des charges

Dans l'ensemble du projet, après débriefing avec Jérôme, les principaux points sont bien respectés. Pendant le développement, plusieurs corrections ont été effectuées pendant le développement pour respecter au mieux le cahier des charges.

Pour le front-end, j'ai pu gagner du temps car les maquettes étaient précises avec les informations données. La mise en forme fût relativement simple avec Bootstrap.

J'ai ressenti principalement des difficultés à coder les contrôleurs des 2 applications car l'implémentation est particulière afin de respecter les mesures de sécurité du produit, et pour correspondre au fonctionnement actuel. Pour cela, Maud m'a été d'une aide précieuse.

Cependant, le cahier des charges a été respecté.

Sur la gestion de projet

Concernant la gestion du projet, nous avons eu quelques jours de retard en raison du travail parallèle. J'ai dû travailler sur différentes fonctions en urgence pour correspondre aux exigences de la certification par l'ANSSI.

De plus, la 1ère semaine du projet, j'ai accompagné SesameIT au FIC (Forum International de la Cybersécurité) à Lille du mardi 28 Janvier 2020 au jeudi 30 janvier 2020. La présence de SesameIT était prévue, mais ma présence s'est confirmée quelques jours seulement avant. Cet évènement a fait glisser légèrement le planning initial, mais pendant le FIC, nous avons discuté du projet ce qui a permis de faire partiellement la partie « Définition et analyse du projet » initialement prévu sur le planning.

Sur l'avenir du projet

Il reste à tester intensément le projet, dont je n'ai que très peu participé en raison des jours qui ont glissés, et la fin de ma période de stage. Cependant, j'ai reçu des retours positifs sur les tests quelques jours après mon départ concernant le projet « Gérer Hash IDS/DPI ».

SesameIT était en attente des résultats d'une partie des tests de l'ANSSI via un testeur indépendant. Malgré aucun résultat officiel à ce moment, les premiers retours sont très positifs ce qui permet de se projeter sur l'avenir de la sonde de détection de façon positive.