

Développement d'une solution de sécurité chez INEO

Année 2023-2024



15 NOVEMBRE 2024

Rédigé par : Enzo Fournet

A l'intention de Luc Castillo, tuteur en entreprise
et Jean-Michel Bruel, tuteur pédagogique

Remerciements

Je tiens, tout d'abord, à exprimer ma profonde gratitude envers toutes les personnes qui ont contribué à mon épanouissement professionnel durant mon alternance.

Je souhaite particulièrement remercier Yannick Marchetaux, mon référent chez INEO, qui m'a offert l'opportunité de travailler sur un projet à la fois formateur et stimulant. Grâce à son soutien, j'ai donc eu la chance de participer à un projet enrichissant qui m'a permis d'apprendre énormément, de gagner en autonomie et de faire les bons choix dans le cadre d'un développement FullStack. Je tiens, également, à le remercier pour sa disponibilité, malgré son emploi du temps chargé, et pour l'aide précieuse qu'il m'a apporté tout au long de mon alternance.

Je souhaite aussi adresser mes remerciements à mon tuteur en entreprise, Luc Castillo, pour son accompagnement constant tout au long de ma mission et de sa disponibilité à répondre à toutes mes questions relatives à la conception et aux besoins de l'application.

J'exprime aussi ma reconnaissance envers tous mes collègues qui m'ont intégré et souvent aidé à mieux comprendre et à apprendre. Mes remerciements s'étendent aussi à Guillaume Piessat pour son accueil chaleureux au sein de l'entreprise.

Table des matières

Remerciements	2
Introduction	5
1 Présentation d'INEO.....	6
1.1 Historique de l'entreprise	6
1.2 Structure de l'entreprise.....	6
1.2.1 Groupe Bouygues	6
1.2.2 Groupe EQUANS.....	7
1.2.3 INEO MPLR	7
1.3 Activité et Clients	8
1.4 Concurrents	10
1.5 Place sur le marché.....	11
1.6 Culture d'entreprise	11
2 Analyse du besoin	12
2.1 Le client et les concernés	12
2.2 Analyse de l'existant	12
2.3 Caractérisation du besoin	14
2.3.1 Présent	14
2.3.2 Futur	14
2.4 Définition des contraintes.....	15
2.5 Définition des objectifs.....	16
2.5.1 Maintien d'une version fonctionnelle	16
2.5.2 Développement de la version 2.0	16
3 Réalisation technique	16
3.1 La méthodologie	16
3.1.1 Outils de conception et de développement	16
3.1.2 Outils de gestion de projet.....	17
3.1.3 Organisation	18
3.2 Le déroulement.....	19
3.2.1 Industrialisation.....	19
3.2.2 Ajustement de la version 1.....	28
3.2.3 Re factoring et passage à Django	31
3.3 Les résultats.....	32
4 Autre Réalisation	33
4.1 DREAM - TEST	33
5 Réflexion sur le rôle de l'IA dans l'apprentissage	34
5.1 Contexte et contraintes professionnelles	34
5.2 Impact	35

5.3	Conclusion	36
6	<i>Bilan</i>	37
6.1	Les objectifs et leur impact	37
6.2	La gestion de projet	37
6.3	Apports personnels	37
	<i>Conclusion</i>	38
	<i>Lexique</i>	39
	<i>Table des illustrations</i>	44
	<i>Annexes</i>	45
	Introduction à Docker, Dockerfile, et Docker Compose	45

Introduction

Dans un paysage numérique où **le coût des cyberattaques s'élève à 129 milliards de dollars en France en 2024**, selon [Statista](#), et où, selon l'[ANSSI](#), **on relève une augmentation de 30% des attaques l'année précédente**, la cyber menace n'a jamais été aussi palpable. **Ces chiffres, plus qu'alarmants, mettent en lumière une réalité incontournable : la cybersécurité n'est plus une option mais une nécessité absolue.**

Dans ce contexte de menace croissante, la mise en place d'une stratégie de sécurité proactive est primordiale. La question se pose alors : Comment une analyse mensuelle et approfondie des failles de sécurité peut-elle non seulement anticiper les risques, mais également renforcer la confiance et la sécurité des clients ? Ce rapport d'alternance propose d'explorer les méthodes et les bénéfices d'une telle approche, en prenant pour exemple l'industrialisation et la refonte d'une application utilisant Angular et Flask, permettant l'analyse des produits fournis aux clients d'INEO.

Au cœur de cette ère numérique, en constante évolution, où les menaces informatiques se multiplient, mon alternance s'est inscrite dans une démarche résolument tournée vers l'avenir. Chargé de reprendre et de transformer une application FullStack Angular et Flask d'analyse des failles, mon objectif était double : non seulement moderniser et optimiser cette plateforme via l'intégration de Docker en industrialisant la solution, mais aussi, et surtout, la refonte complète de l'application.

À travers ce rapport, je relaterai le processus d'industrialisation et de refonte que j'ai mené sur l'application Gestion Faille, illustrant comment une attention minutieuse portée à la sécurité peut non seulement prévenir les risques, mais également contribuer à construire un environnement numérique plus sûr pour tous.

1 Présentation d'INEO

1.1 Historique de l'entreprise

INEO fait aujourd'hui partie de la filiale EQUANS Digital du groupe Bouygues, initialement créée en 2001. Elle a rejoint le groupe Engie Solutions en 2020, avant de fusionner récemment avec TINEA, une autre entreprise du groupe EQUANS Digital. Cette évolution fait suite au rachat par Bouygues, marquant une étape importante dans l'expansion de l'entreprise au sein du secteur digital.

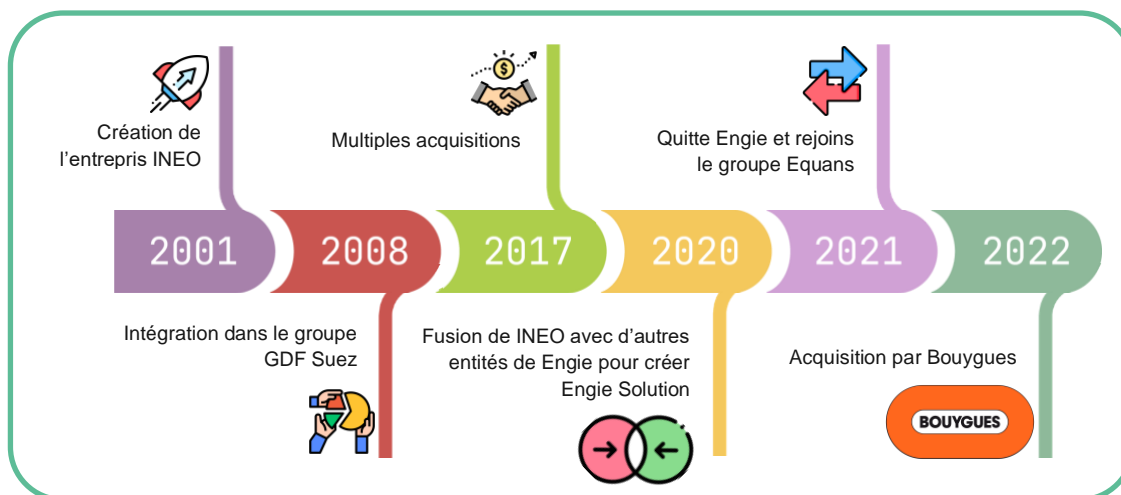


Figure 1 : Frise chronologique de l'historique d'INEO

1.2 Structure de l'entreprise

1.2.1 Groupe Bouygues

Comme mentionné précédemment, INEO et ses homologues de Engie Solutions ont été acquis par Bouygues pour former une nouvelle filiale digitale au sein du GROUPE EQUANS. INEO fait donc partie de la filiale Énergie et Service représenté par EQUANS dans le groupe Bouygues.

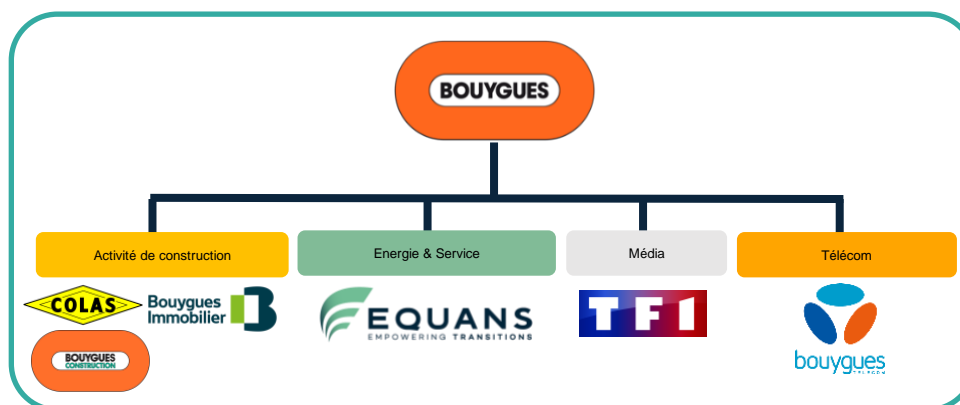


Figure 2 : Organigramme du groupe Bouygues

1.2.2 Groupe EQUANS

Comme nous pouvons le voir ci-dessous, INEO fait parti de la filiale digitale du groupe EQUANS. Dans laquelle se trouve également TINEA, une entité avec laquelle INEO a fusionné en 2024.

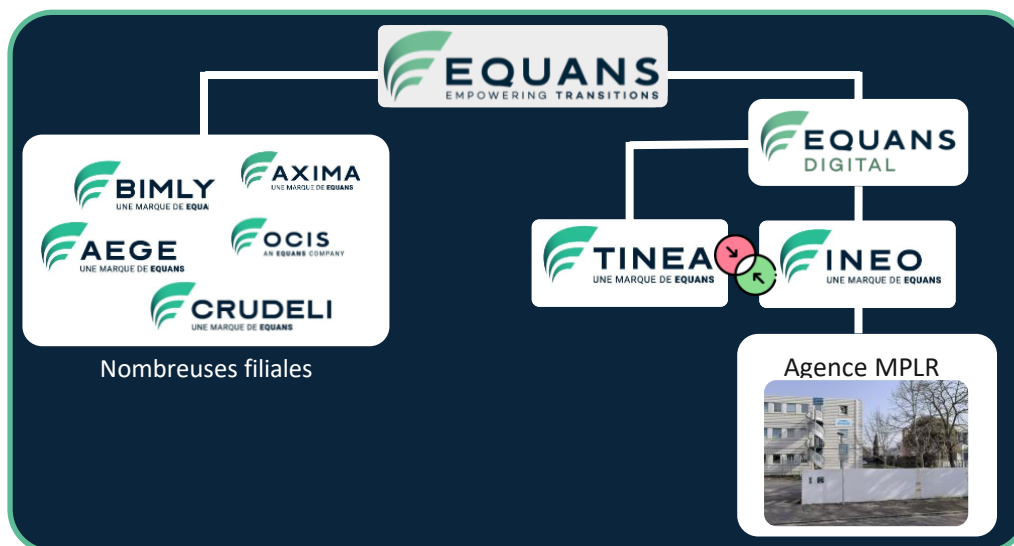


Figure 3 : Organigramme du groupe EQUANS

1.2.3 INEO MPLR

Parlons maintenant de l'agence dans laquelle j'ai été intégré, l'agence INEO MPLR (Midi Pyrénéen Languedoc Roussillon) située à Toulouse, près du Mirail. Cette agence regroupe le pôle data et le pôle industriel d'INEO. Ci-dessous, vous trouverez un organigramme simplifié de l'agence MPLR où figurent Yannick MARCHETAUX, mon référent, et Luc CASTILLO, mon tuteur en entreprise.

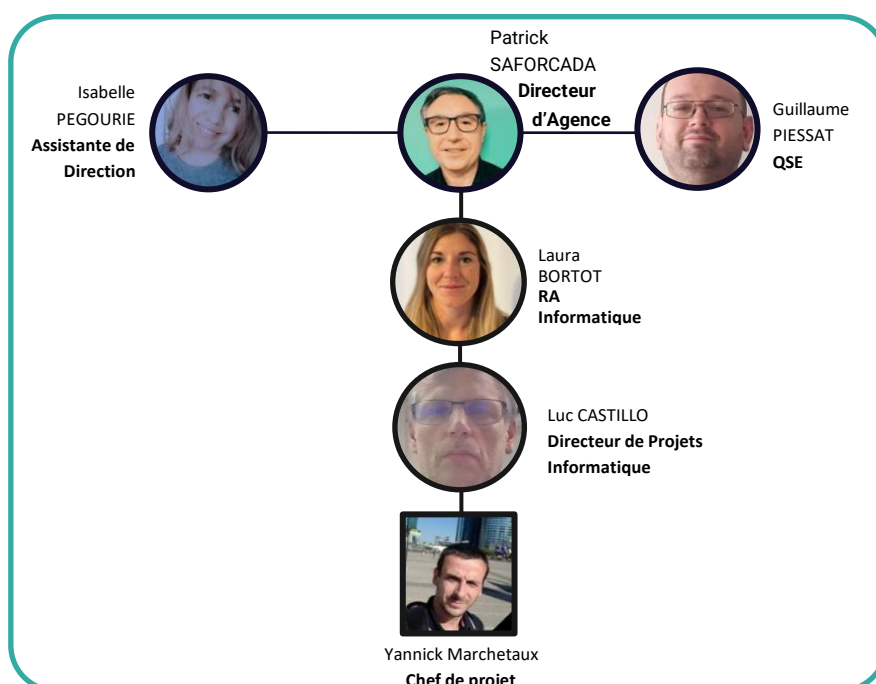


Figure 4 : Organigramme simplifié de l'agence INEO MPLR

1.3 Activité et Clients

INEO a eu l'occasion de collaborer avec un éventail impressionnant d'entreprises et de grands groupes, parmi lesquels figurent des acteurs majeurs tels que NAVAL Group, la SNCF et TOTAL.

Ci-dessous, vous trouverez une liste non exhaustive des clients d'INEO :



Figure 5 : Liste non exhaustive des clients d'INEO

Pour illustrer concrètement le travail effectué par INEO, je vais brièvement présenter un projet dirigé par Yannick, mon référent. Il s'agit du projet Tahiti, développé dans le cadre d'un contrat avec Électricité de Tahiti (EDT). L'objectif de ce projet était de collecter des données de production incluant le mix énergétique de l'île ainsi que les données de consommation pour assister les dispatcheurs dans l'équilibrage du réseau électrique en fonction des diverses sources d'énergie disponibles, telles que l'hydroélectricité, la production thermique et l'énergie solaire.

Voici des exemples d'interfaces développées dans le cadre de ce projet :



Figure 6 : Exemple d'interface de visualisation des données du projet Tahiti

Ce projet est un exemple parfait de l'expertise d'INEO, tant dans le secteur de l'énergie que dans le développement d'applications efficaces répondant précisément aux besoins du client.

1.4 Concurrents

Dans la région toulousaine, plusieurs entreprises pourraient être considérées comme des concurrents d'INEO, surtout dans les domaines du conseil en informatique et de la cybersécurité. En plus de Capgemini et Sopra Steria, vous pouvez également prendre en compte Atos et CGI :

- **Atos** : Cette entreprise opère à l'échelle mondiale et propose une gamme complète de services informatiques, y compris des solutions de cybersécurité. Elle est réputée pour ses services numériques étendus et a une présence forte en Europe.
- **CGI** : Il s'agit d'une autre grande entreprise de services informatiques et de conseil qui offre des services intégrés en TI et des processus d'affaires, incluant la cybersécurité et l'intégration de systèmes. CGI est comparable à Sopra Steria et Capgemini en termes d'offres et d'engagements clients.
- **Capgemini** : Bien connu dans le domaine de la consultance en technologie, Capgemini fournit des services en ingénierie et en technologie, y compris la cybersécurité, essentiels pour rivaliser sur le marché de la haute technologie.
- **Sopra Steria** : Spécialisée dans la consultation IT, cette entreprise offre également des solutions intégrées qui couvrent la cybersécurité, faisant d'elle un concurrent direct dans le domaine des services technologiques et de la cybersécurité.

Ces entreprises, comme INEO, mettent un fort accent sur la technologie et la cybersécurité, ce qui en fait des concurrents directs dans la région.

1.5 Place sur le marché

EQUANS, avec un chiffre d'affaires de 13 milliards d'euros, s'impose comme un acteur majeur sur le marché international. L'entreprise emploie environ 97 000 collaborateurs répartis dans 20 pays, ce qui témoigne de son envergure et de sa capacité à gérer des projets d'une grande complexité à l'échelle mondiale.

La réputation d'EQUANS est également renforcée par sa collaboration avec de grandes entreprises et institutions. Parmi ses clients, on retrouve des noms prestigieux tels que Bouygues Immobilier, la SNCF, NAVAL Group, SANOFI, TOTAL, la RATP, VINCI, ENGIE, et RTE. Ces collaborations illustrent non seulement la confiance que ces grandes entités placent en EQUANS, mais aussi son expertise dans des secteurs variés allant de l'immobilier et le transport à l'énergie et la santé.

Ces partenariats stratégiques permettent à EQUANS de maintenir une position de leader dans le domaine des services multi-techniques, en particulier dans les secteurs où les exigences en matière de technologie et de sécurité sont élevées. Cette diversité de clients et de secteurs d'activité contribue à la stabilité et à la croissance continue de l'entreprise sur le marché global.

1.6 Culture d'entreprise

Chez INEO, la cohésion au sein des équipes est primordiale pour assurer l'efficacité du travail collaboratif. C'est pourquoi des repas de cohésion sont régulièrement organisés, favorisant ainsi les échanges informels et renforçant le lien entre collègues. De plus, l'entreprise adopte des horaires flexibles permettant à chacun de s'organiser selon ses besoins personnels, ce qui contribue à un meilleur équilibre entre vie professionnelle et vie privée.

Le télétravail est également une composante intégrale de l'organisation d'INEO, offrant aux employés la possibilité de travailler dans un environnement qui leur est confortable et adapté, tout en maintenant la productivité et l'engagement envers les objectifs de l'entreprise.

Il est courant, lors de l'anniversaire d'un employé, que celui-ci apporte des viennoiseries à partager avec l'équipe, instaurant un moment convivial et festif qui renforce le sentiment d'appartenance et la culture de partage de l'entreprise.

Les réunions quotidiennes, ou dayli, sont également un aspect important de la structure de travail chez INEO; dans mon cas, elles se tiennent tous les lundis à 14h avec Yannick et Luc. Ces réunions permettent de faire le point sur les projets en cours, d'ajuster les plans d'action et de s'assurer que tous les membres de l'équipe sont alignés sur les mêmes objectifs.

2 Analyse du besoin

2.1 Le client et les concernés

Actuellement, les principales parties concernées par l'application Gestion Faille sont les équipes internes d'INEO, qui effectuent jusqu'à présent l'analyse des failles de sécurité de leurs applications de manière manuelle. À terme, cet outil leur permettra de gagner un temps précieux dans la livraison de leurs produits. En outre, nos clients sont également directement concernés, puisque l'application cherche à répondre à une problématique de plus en plus préoccupante : garantir la sécurité des produits vendus à nos clients ou utilisés en interne. C'est pour cette raison que des clauses relatives à l'analyse des failles ont été intégrées dans les contrats signés avec les clients d'INEO. Les clients bénéficieront donc directement de ces analyses et de leur efficacité.

2.2 Analyse de l'existant

À mon arrivée chez INEO, un embryon d'application avait été élaboré par un ancien stagiaire, Noé Faucher. Durant son stage de 12 semaines, Noé avait initié le développement d'un outil de gestion de vulnérabilités, destiné à suivre les failles de sécurité des différents produits maintenus par l'entreprise. Sa contribution avait posé les fondements de l'importation de données concernant les packages présents dans les différentes versions d'un produit et de la recherche de vulnérabilités au sein de ces packages. Les données collectées étaient structurées pour refléter les relations entre produits, versions, packages et vulnérabilités correspondantes, permettant ainsi la génération de divers rapports essentiels à l'amélioration de la sécurité des produits. Durant son passage, Noé a également implanté un mécanisme permettant la mise à jour régulière des informations sur les vulnérabilités qui permet aux opérateurs d'analyser et d'ajuster la sévérité des vulnérabilités selon le contexte d'utilisation du package. Il a mis en place une interface initiale et a travaillé sur la documentation pour assurer la maintenabilité de l'outil.

Le schéma ci-dessous illustre l'état de l'architecture de l'application telle que développée par Noé Faucher durant son stage. Noé avait mis en place une API conçue avec Flask, un framework Python spécialisé dans la création d'APIs, qui était connectée directement à une base de données MariaDB. Cette connexion était établie par le biais de requêtes spécifiques qu'il avait programmées lui-même. De plus, un frontend utilisant Angular permettait d'interagir avec l'API de l'application. Chaque composant de l'application, dans cette architecture, était utilisé directement sur la machine hôte et exécuté localement.

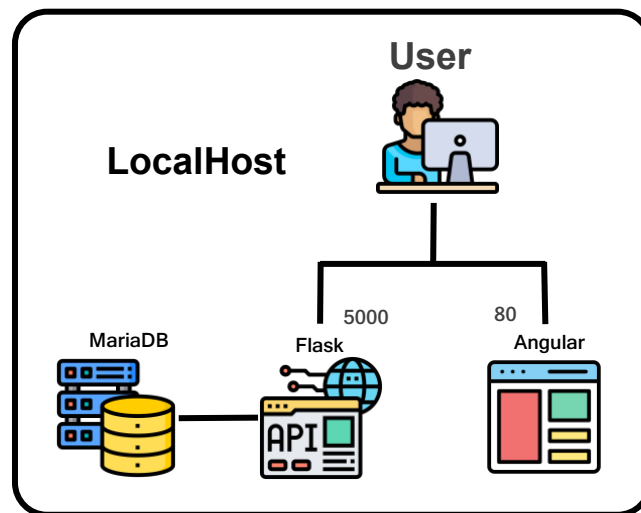


Figure 7 : Représentation de l'application réalisé par Noe Fauche

2.3 Caractérisation du besoin

Pour cerner précisément les besoins, il est crucial de différencier les attentes immédiates, celles à court et moyen terme, concernant l'application Gestion Faille, et les projections à long terme.

2.3.1 Présent

Actuellement, INEO est tenu de fournir à ses clients des rapports de vulnérabilité clairs et précis, conformément aux contrats établis qui stipulent, comme mentionné précédemment, des clauses relatives à l'analyse des vulnérabilités. Ces rapports doivent non seulement répertorier les vulnérabilités identifiées sur les produits mais également inclure une réévaluation faite par le mainteneur, qui réévalue la sévérité de chaque faille dans le contexte spécifique du produit, accompagnée d'une brève explication. L'application doit donc être en mesure de générer ces rapports détaillés pour les clients.

Ci-dessous un digramme représentant les cas d'usages représentant les besoins actuels.

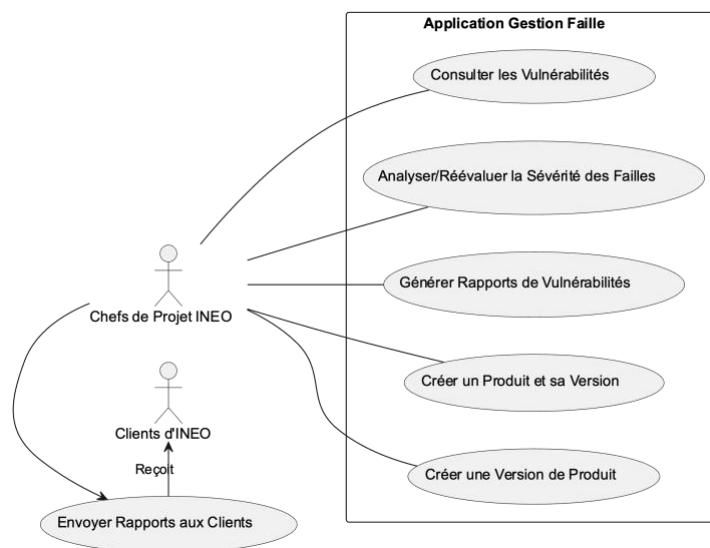


Figure 8 : Diagramme de cas d'utilisation de l'application Gestion Faille

2.3.2 Futur

À plus long terme, l'application devra gagner en modularité pour s'adapter aux nouveaux contrats et aux particularités de certains produits. Plus important encore, elle devra être optimisée pour assurer la viabilité, la rapidité, et surtout l'efficacité de l'analyse et de la réévaluation des vulnérabilités découvertes sur les produits. Cela implique de pouvoir communiquer rapidement les informations relatives aux vulnérabilités et de proposer des correctifs dans les meilleurs délais. Une telle amélioration garantira l'efficacité de ce processus.

2.4 Définition des contraintes

Plusieurs contraintes ont été définies dans le cadre du développement de l'application GestionFaille. Premièrement, l'application devra être industrialisée via Docker afin de garantir un contrôle et un versionnage de l'environnement de développement et de déploiement, ce qui facilitera grandement le développement et le déploiement de l'application. De plus, le scan de vulnérabilité des produits devra être effectué à l'aide de l'outil OSV Scanner et de sa base de données. Il sera donc nécessaire de générer des SBOMs pour nos différents produits afin de permettre leur analyse par l'OSV Scanner, qui accepte uniquement les fichiers de type SBOM en entrée.

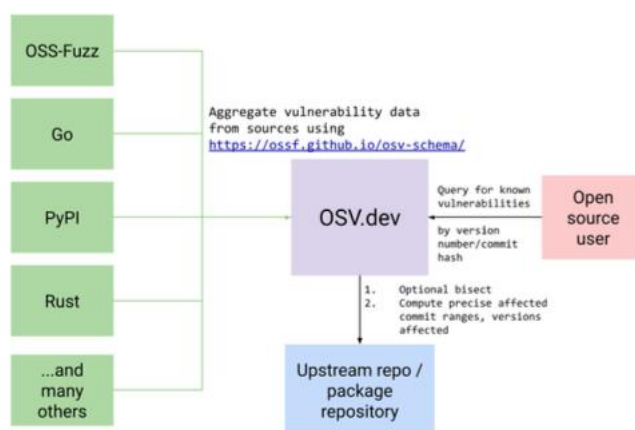


Figure 9 : Schéma de la base de données OSV

Ensuite, les comptes rendus devront être générés dans un format facilement lisible, tel que l'HTML ou le PDF.

2.5 Définition des objectifs

2.5.1 Maintien d'une version fonctionnelle

Dans un premier temps, l'objectif est d'industrialiser et de rendre utilisable la version actuelle de l'application. Cela signifie que je dois retirer l'ancien environnement virtuel Python et les applications exécutées en local pour les transférer dans des conteneurs Docker. Le deuxième objectif est de maintenir cette version, ce qui implique que lorsque un bug est rencontré, je dois être capable de le régler ou d'apporter des ajustements sur demande.

2.5.2 Développement de la version 2.0

L'objectif suivant consiste d'abord à refactoriser toute la partie backend de l'application en remplaçant le framework Flask par Django, car ce dernier est mieux adapté pour gérer les modèles de données complexes et accélère le développement d'applications robustes. Une fois le backend et l'API renforcés, je pourrai alors commencer à améliorer le frontend pour offrir une interface plus simple et intuitive à utiliser et enrichie de nouvelles fonctionnalités.

3 Réalisation technique

3.1 La méthodologie

3.1.1 Outils de conception et de développement

Pour la conception, j'ai utilisé Draw.io, un logiciel qui permet de réaliser facilement des schémas, notamment pour la conception de ma base de données. J'ai également utilisé PlantUML pour élaborer quelques diagrammes de cas d'utilisation.

En ce qui concerne le développement, j'ai principalement utilisé Docker avec Docker Desktop, une interface facilitant l'usage de Docker sur Windows et MacOS. Pour le développement proprement dit, j'ai employé Visual Studio Code et Postman un outil qui simplifie l'envoi de requêtes HTTP, très utile notamment pour le développement d'API. Enfin, j'ai utilisé MobaXterm pour me connecter à mes machines de test et de production, afin de tester l'application ou de la déployer sur le réseau de l'entreprise.

J'ai eu la chance de travailler avec des outils que je connaissais déjà bien, ce qui m'a permis de les prendre en main rapidement et facilement.

3.1.2 Outils de gestion de projet

En ce qui concerne la gestion de projet, nous utilisons GitLab, une plateforme Git qui permet non seulement de versionner le code et de définir les versions de l'application, mais aussi de gérer les tickets en ouvrant des issues relatives au travail à réaliser. Chez INEO, le travail associé à une issue doit être réalisé sur une branche GIT spécifique nommée selon la convention suivante : <numéro de l'issue>-<titre de la branche>-<trigramme de l'utilisateur>. Par exemple, **24-Refactoring-eft** où « 24 » est le numéro de l'issue, « Refactoring » le titre, et « eft » le trigramme correspondant à Enzo Fournet.

Ensuite, les commits doivent eux aussi suivre des conventions précises. Un commit doit toujours être formaté comme suit : refs #<numéro de l'issue> - <titre du commit>, suivi d'une description du travail réalisé dans le commentaire accompagnant le commit. Par exemple : **refs #24 – début refactoring** où #24 est le numéro de l'issue et « début refactoring » le titre du commit.

Lorsque le travail sur une branche est terminé, je procède alors à une merge request qui sera ensuite traitée par Yannick. Une merge request est une demande pour intégrer les modifications réalisées dans la branche principale du projet.

Voici un exemple de merge request :

The screenshot displays a GitLab Merge Request (MR) interface. The main heading is "Resolve "Fix - Correction de l'apparition de deux lettre v dans le PURL"". Below this, it indicates the MR was "Merged" and "Opened 1 week ago by enzo fournnet". The title "Resolve "Fix - Correction de l'apparition de deux lettre v dans le PURL"" is prominently displayed. The interface shows the MR was merged by Yannick Marchetaux 6 days ago. The right sidebar provides additional details: Assignee (Yannick Marchetaux), Milestone (Version 1.2.2), Labels (Anomalie, Code), and Lock merge request (Unlocked). The bottom section shows a list of activities: enzo fournnet changed milestone to %Version 1.2.2 1 week ago, enzo fournnet added labels 1 week ago, and Yannick Marchetaux merged 6 days ago.

Figure 10: Exemple de merge request sur GitLab

3.1.3 Organisation

En ce qui concerne l'organisation et la méthode de travail, aucune contrainte spécifique ne m'a été imposé. J'ai donc abordé les missions qui m'étaient confiées en adoptant une approche séquentielle, en commençant par les tâches les plus importantes. Aucune durée de travail ni deadline précise n'étant définie, cette méthode me permet de travailler à mon rythme, de prendre le temps d'apprendre et de faire les bons choix pour les tâches qui le permettent.

Voici tout de même un diagramme de Gantt représentant mon travail sur les semaine passé en entreprise :

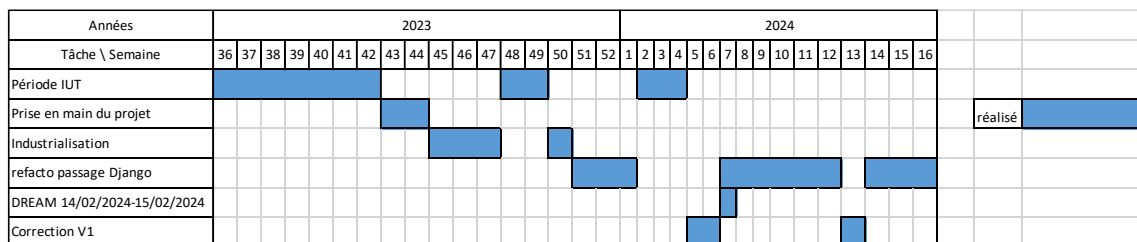


Figure 11 : Gantt réalisé

3.2 Le déroulement

Veillez noter que cette section du rapport contient de nombreuses explications techniques et fait référence à Docker. Je vous recommande donc de lire [l'annexe dédiée à Docker](#) et de consulter [le lexique](#) pour une meilleure compréhension des termes employés.

Tout d'abord, un peu de contexte : lorsque je mentionnerai la version 1 de l'application, il s'agira de la version actuellement maintenue sur Flask et industrialisée via Docker. Quant à la version 2, il s'agira de celle en cours de refactoring, où nous changeons de framework backend.

3.2.1 Industrialisation

Lors de mon arrivée chez INEO, j'ai récupéré la version de l'application développée par Noé Faucher. J'ai alors été chargé d'industrialiser l'application réalisée par Noé. Il m'a donc fallu me plonger dans le code et dans la conception de l'application pour comprendre ce qui avait été fait, étant donné que la documentation manquait beaucoup de détails, afin de transférer chaque composant applicatif dans son propre conteneur Docker.

3.2.1.1 Le Back-End

J'ai donc commencé par le backend. Le backend avait été mis en place dans un environnement virtuel Python, ce qui permettait à l'API de s'exécuter au sein de son propre environnement en utilisant ses propres dépendances. Cela représente un avantage pour la suite, car le principe de conteneurisation que j'ai appliqué dans cette phase d'industrialisation est similaire à ce qui peut être fait avec un environnement virtuel Python. Puisque le principe d'un environnement Python est de réduire les dépendances du programme vis-à-vis de son hôte afin de faciliter le portage sur d'autres machines ou hôtes, Docker représente une solution encore plus robuste. En effet, Docker permet non seulement de gérer les dépendances de manière isolée, mais aussi d'encapsuler l'application dans des conteneurs qui peuvent être exécutés de manière cohérente sur n'importe quel système disposant de Docker. Cela évite les problèmes courants liés aux différences d'environnements entre le développement et la production, rendant les applications beaucoup plus portables et faciles à déployer. C'est pourquoi l'adoption de Docker était une étape logique dans l'industrialisation de notre application.

Ci-dessous le DockerFile du BackEnd :

```
FROM alpine:latest

# Récupérer la version d'Alpine, transformer la version dans la version correct, et la stocker dans une variable

RUN VERSION_REPOSITORY_ALPINE=$(cat /etc/alpine-release | awk -F'.' '{print $1"."$2}') && \

    echo "http://192.168.200.24:8081/repository/alpine/v${VERSION_REPOSITORY_ALPINE}/main" > /etc/apk/repositories && \

    echo "http://192.168.200.24:8081/repository/alpine/v${VERSION_REPOSITORY_ALPINE}/community" >> /etc/apk/repositories

# Installation des paquets nécessaires

RUN apk update && apk add --no-cache python3 py3-pip mariadb-connector-c-dev build-base python3-dev openssl ca-certificates tzdata dcron curl

# Définir la timezone

RUN cp /usr/share/zoneinfo/Europe/Paris /etc/localtime && \

    echo "Europe/Paris" > /etc/timezone

# Installation des certificats pour l'api de l'OSV-SCAN

RUN echo | openssl s_client -connect api.osv.dev:443 2>/dev/null | openssl x509 > /etc/ssl/certs/osv.crt

RUN update-ca-certificates

# Copie du contenu de backend

COPY ./back/backend/ /back-end/

# Copie du crontab

COPY ./back/crontab_py /etc/crontabs/root

#Création du répertoire d'upload

RUN mkdir -p /back-end/python_src/src/api/upload/

# Définir le répertoire de travail

WORKDIR /back-end/python_src/

# Installation des dépendances

RUN pip install --break-system-pac -r requirements.txt

# Définir les droits sur le répertoire de travail

RUN chmod -R 777 /back-end/python_src

# Permet de garder le container UP sans lancer runApi.sh utile pour debug manuellement le conteneur

# il est pertinent de lancer le container avec cette command pour du dev

# CMD [ "tail", "-f", "/dev/null" ]

CMD ["sh", "./runApi.sh", "-r"]
```

Figure 12 : DockerFile du BackEnd en version V1

Dans ce Dockerfile, nous débutons par la modification des dépôts du conteneur pour utiliser ceux du proxy de INEO. Ce processus implique l'extraction de la version actuelle d'Alpine Linux et sa transformation pour correspondre aux versions hébergées par le proxy d'INEO. Cette modification assure que toutes les installations ultérieures de packages se feront via ces dépôts spécifiques.

Ensuite, le Dockerfile procède à l'installation des bibliothèques nécessaires qui incluent Python, divers outils de développement, et des certificats pour sécuriser les connexions. Ces installations préparent le terrain pour toute fonctionnalité requise par l'application.

La définition de la timezone à 'Europe/Paris' est cruciale pour garantir que toutes les opérations liées à la date et l'heure soient cohérentes, en particulier lors de la génération de rapports.

L'ajout du certificat pour l'API OSV-SCAN est une étape clé pour permettre au conteneur d'interagir de manière sécurisée avec l'API OSV externe, un service essentiel pour le scanner de vulnérabilités.

Le Dockerfile continue par la copie du code nécessaire au fonctionnement de l'application backend et des fichiers de configuration comme le crontab, qui est utilisé pour planifier des tâches régulières. Un répertoire spécifique pour l'upload est également créé, démontrant une organisation et une préparation des espaces de travail nécessaires.

Le répertoire de travail est défini dans une sous-section du backend pour centraliser toutes les opérations de déploiement et d'exécution du code dans ce lieu précis. Cela est suivi de l'installation des dépendances Python, qui sont essentielles pour le fonctionnement des scripts et applications dans le conteneur.

Des droits d'exécution sont attribués au répertoire de travail pour permettre une manipulation flexible des fichiers par divers processus. Finalement, le Dockerfile termine par la définition de la commande par défaut qui exécute un script pour lancer l'API, tout en permettant de maintenir le conteneur actif pour le débogage ou le développement ultérieur.

Le backend est désormais opérationnel mais ne fonctionne pas correctement car il lui manque sa base de données, sans laquelle il ne peut rien sauvegarder et n'est donc pas fonctionnel. Concernant la base de données, la mise en place a été très simple car, pour certaines applications, les entreprises ou entités qui en assurent le développement proposent des images Docker prêtes à être utilisées.

3.2.1.2 La base de données

Pour intégrer la base de données dans notre environnement Docker, j'ai utilisé l'image officielle de MariaDB. Puis j'ai créé un Dockerfile pour personnaliser l'image en y ajoutant le script SQL nécessaire à la création de la structure de la base de données.

Voici le Dockerfile correspondant à la base de données :

```
FROM mariadb:latest  
  
# Copier le script .sql dans le dossier d'initialisation de MariaDB  
  
COPY ./back/backend/db/dataBase.sql /docker-entrypoint-initdb.d/dataBase.sql
```

Figure 13 : DockerFile de la base de données

Dans ce Dockerfile, l'instruction COPY est utilisée pour placer notre script SQL dans un répertoire spécial reconnu par l'image officielle de MariaDB. Les scripts dans ce dossier sont exécutés automatiquement au premier démarrage du conteneur, ce qui permet d'initialiser la base de données avec la structure et les données nécessaires. C'est une méthode efficace pour automatiser la configuration de la base de données lors de la mise en place de l'environnement avec Docker.

Pour garantir une communication efficace entre les conteneurs, j'ai exploité les fonctionnalités de mise en réseau de Docker. Celles-ci offrent la possibilité de créer des réseaux internes qui permettent aux conteneurs de se connecter les uns aux autres de manière sécurisée. Cette configuration est essentielle pour assurer une intégration sans accroc des différents services de notre application. Ainsi, j'ai établi un réseau Docker au sein du fichier docker-compose de l'application, regroupant à la fois le backend et la base de données, et leur attribuant des adresses IP fixes pour faciliter leur communication directe.

3.2.1.3 Le Front-End

L'étape suivante concerne le frontend, qui se complexifie légèrement puisque le frontend, construit avec Angular, nécessite d'être compilé avant utilisation. Yannick, mon référent technique, m'a alors fourni un exemple de Dockerfile adapté à ces besoins, me permettant ainsi d'orchestrer le processus de construction du frontend.

Voila ci-dessous le DockerFile du Front que je vais vous décrire :

```
FROM node:18.16-alpine AS build

WORKDIR /build

# Récupération des sources et installation des dépendances
COPY ./front/frontend /build

RUN npm install --force

# Build de l'application
RUN npm run build

# Création de l'image de release
FROM nginx:1.15.2-alpine as release

# Copie des fichiers de configuration et de l'application depuis le build
WORKDIR /usr/share/nginx/html

COPY --from=build /build/dist/frontend /usr/share/nginx/html

COPY ./front/nginx.conf /etc/nginx/nginx.conf

COPY ./front/startup.sh /

CMD ["sh", "/startup.sh"]
```

Figure 14 : DockerFile du FrontEnd en Version 1

Le Dockerfile pour le frontend Angular est donc un peu plus complexe que celui utilisé pour le backend ou la base de données, principalement parce qu'il implique deux étapes distinctes dans le processus de conteneurisation. Le Dockerfile est structuré en utilisant la technique de multi-stage build, qui est idéale pour créer des images légères en production.

Dans la première étape, nommée *build*, l'image de base *node:18.16-alpine* est utilisée pour fournir l'environnement Node.js nécessaire pour construire l'application Angular. La légèreté de l'image Alpine est un choix délibéré pour réduire la taille de l'image finale. Tous les fichiers sources nécessaires sont copiés dans un répertoire de travail */build* et les dépendances du projet sont installées via *npm install --force*. Une fois les dépendances installées, la commande *npm run build* compile l'application en une version optimisée pour la production.

La seconde étape, *release*, commence par instancier une nouvelle image à partir de *nginx:1.15.2-alpine*, qui est un serveur web léger et performant, également basé sur Alpine Linux. Ce serveur sera responsable de la livraison des fichiers statiques de l'application Angular. Le contenu compilé lors de la première étape est copié dans le répertoire */usr/share/nginx/html* de l'image, ce qui est le répertoire par défaut pour les fichiers statiques servis par Nginx. De plus, une configuration personnalisée de Nginx est placée dans */etc/nginx/nginx.conf* et un script de démarrage est copié à la racine.

Le script de démarrage, exécuté via la commande *CMD*, initie le serveur Nginx et peut inclure des commandes supplémentaires nécessaires pour configurer l'environnement de production au démarrage du conteneur.

Ce Dockerfile est une illustration parfaite de l'efficacité des builds multi-étapes dans Docker, permettant de séparer les étapes de construction et de déploiement de l'application, afin de produire une image finale aussi petite et optimisée que possible.

3.2.1.4 L'arborescence et loganisation du code

Concernant l'organisation du code, j'ai choisi d'établir une structure cohérente, regroupant l'ensemble des composants au sein d'un même dépôt Git (repository). Cette structuration facilite la compréhension, le développement et la maintenance du projet.

Le fichier *README.md* à la racine du projet sert de point d'entrée à la documentation. Il contient une partie de la documentation, bien que celle-ci ne soit pas complètement à jour dans la version actuelle du projet. Il est prévu que la version 2 remplace cette documentation initiale.

Le répertoire *deployment* inclut le fichier *docker-compose.yml*, qui orchestre la composition des services de l'application, incluant les trois services principaux (Back-End, Front-End, Base de données) ainsi que la configuration réseau nécessaire à la communication entre conteneurs. Nous détaillerons cette composition et son rôle à l'aide d'un schéma explicatif ultérieurement.

Le répertoire back contient le Dockerfile spécifique au backend ainsi que les sources du backend, y compris la documentation de l'API (api-doc.yaml), les configurations de base de données, le script SQL initial (init.sql), et le code source Python dans python_src. Des scripts pour l'exécution de l'API et les tests sont également présents, ainsi qu'une tâche cron dans crontab_py.

Enfin, le répertoire front est consacré au frontend avec son propre Dockerfile. Il comprend le code source Angular du frontend, les configurations TypeScript (tsconfig.*.json), ainsi que les fichiers nécessaires à la configuration de Nginx (nginx.conf) et au script de démarrage (startup.sh).

Cette structure est pensée pour favoriser la modularité et l'autonomie de chaque composant du système. Chaque répertoire peut être considéré comme un module indépendant, ayant tout ce qui est nécessaire pour sa construction, son déploiement, et son fonctionnement autonome, tout en permettant une intégration harmonieuse avec les autres composants via Docker Compose.

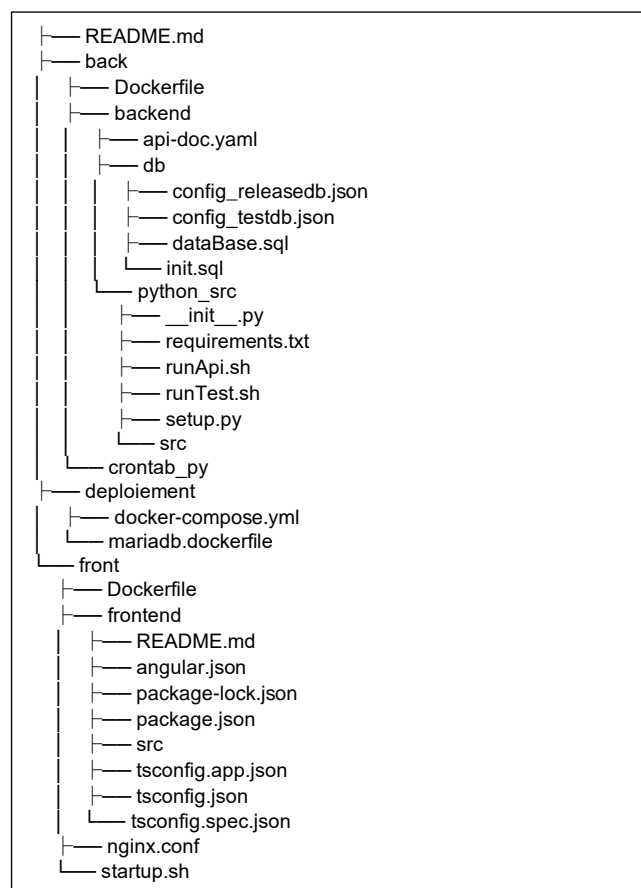


Figure 15 : Arborescence du projet en Version 1

Avec cette organisation, l'objectif est d'avoir un système facile à naviguer et à comprendre pour les nouveaux développeurs rejoignant le projet, et également de simplifier les mises à jour futures du système.

3.2.1.5 Schéma de l'application

- Schéma de la base de données :



Figure 16 : Schéma de base de données en Version 1

- Schéma de l'application :

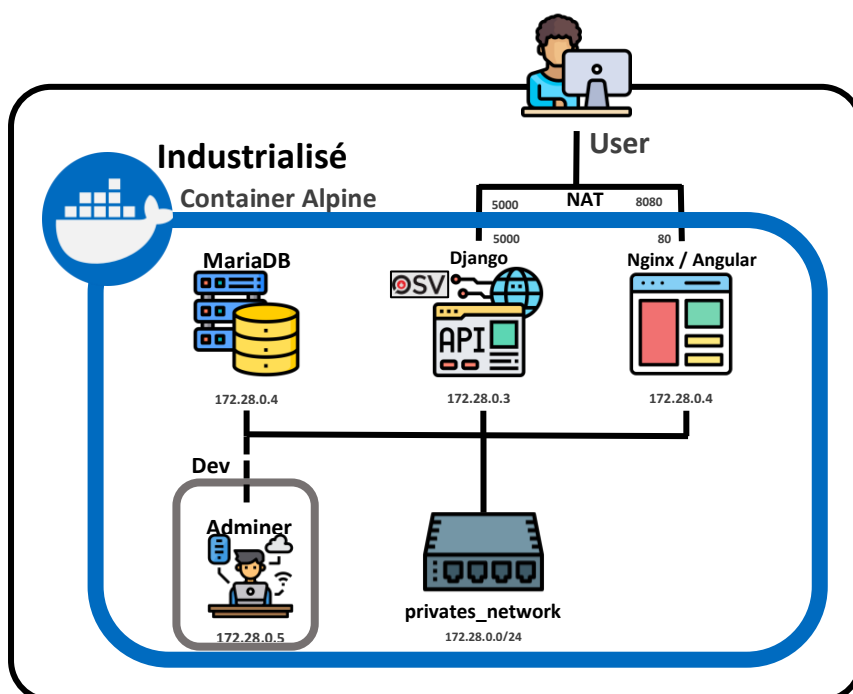


Figure 17 : Schéma représentatif de l'app en Version 1

3.2.1.6 Versioning de l'application

L'infrastructure d'INEO comprend un proxy qui permet de créer des dépôts de paquets ou même d'images Docker. Pour déployer une version de l'application au sein de l'infrastructure d'INEO et la rendre opérationnelle, il est donc nécessaire de construire les images à partir du fichier docker-compose du projet. Ces images sont ensuite envoyées sur le proxy, ce qui permet leur accès depuis les machines de production qui n'ont pas de connexion directe à Internet.

Lorsqu'une nouvelle version est taguée dans GitLab, c'est-à-dire lorsqu'une version est attribuée à un commit, il est essentiel de mettre à jour les numéros de version dans le fichier docker-compose. Cela permet de construire l'application avec la version courante et de pousser la bonne version sur le proxy pour une utilisation en production.

3.2.1.7 Synthèse

L'industrialisation de l'application chez INEO a impliqué une série d'étapes méthodiques pour encapsuler les différents composants de l'application dans des conteneurs Docker, assurant ainsi leur portabilité et leur efficacité opérationnelle au sein de l'infrastructure existante. Le processus a débuté par l'intégration du backend dans un environnement virtuel Python, bénéficiant de l'isolation des dépendances, similaire à celle offerte par Docker, mais avec une portabilité accrue. En parallèle, l'adoption d'une image officielle de MariaDB a simplifié l'initialisation de la base de données en y ajoutant le script SQL via un Dockerfile spécifique.

Le frontend, développé avec Angular, a requis une stratégie de construction en deux étapes, où le code source est d'abord compilé dans un environnement Node.js basé sur Alpine Linux pour sa légèreté, puis intégré dans une image Nginx pour la mise en production. Le script de démarrage inclut dans cette image finale assure le lancement et la configuration du serveur web.

En termes d'organisation, une arborescence cohérente a été adoptée pour le dépôt Git, avec une documentation initiale dans le README.md et une séparation claire des composants dans des répertoires dédiés (back, déploiement, et front). Le fichier docker-compose.yml centralise la configuration des services, la gestion des réseaux et la version des images, facilitant ainsi le déploiement et la maintenance.

Enfin, le versioning et le déploiement de l'application sont gérés via un proxy interne à l'entreprise, qui permet de construire et d'héberger les images Docker nécessaires. L'update des numéros de version dans le fichier docker-compose.yml coïncide avec la création de tags dans GitLab, synchronisant ainsi le déploiement des versions dans l'environnement de production.

Cet effort d'industrialisation vise à rationaliser le développement, le test et la mise en production de l'application, tout en préparant le terrain pour des maintenances futures et l'intégration de nouveaux développeurs au projet.

3.2.2 Ajustement de la version 1

3.2.2.1 Correctif rapport

Lors de l'utilisation de l'application, il m'a été rapporté que parfois deux lettres « v » apparaissaient au niveau de la colonne « package » dans le rapport. Pour remédier à ce problème, j'ai créé une issue correspondante dans notre GitLab. Ensuite, j'ai travaillé à la résolution du bug dans une branche dédiée à cet effet. Une fois le correctif apporté, j'ai réalisé une merge request pour que les modifications soient intégrées dans la prochaine version de l'application.

Avant modification :

```
html += "<tr><td>" + package["purl"].replace("pkg:", "").replace("@", " v")
```

Paquets et Vulnérabilités	
Package	Liste des vulnérabilités
golang/github.com/cloudflare/circl vv1.3.3	Date de création et de dernière modification
	08/01/2024 16:45:05
	/
	19/01/2024 00:11:35

Figure 18 : Illustration du problème de " vv " dans le rapport

Après modification :

```
html += "<tr><td>" + package["purl"].replace("pkg:", "").replace("@", " v").replace("vv", "v")
```

Comme on peut le constater il s'agit ici d'ajustement très simple à réaliser.

Il a également été nécessaire de renommer certaines colonnes qui ne correspondaient pas aux attentes des chefs de projet ayant testé l'application.

Sévérité	Analyse par le mainteneur			CIRCL
	Date de l'analyse	Réévaluation de la sévérité	Détails de l'analyse	
	Aucune réévaluation			
	Analyse par le mainteneur			

Figure 19 : Colonnes renommées dans le rapport

Ci-dessous un exemple de rapport générer :

<div>  </div> <div>Rapport mensuel sur les vulnérabilités - 16 Avril 2024</div> <div>OSV SCANNER v1.4.2</div>				
Produit : CDM • Version : 2.6.0 - Information produit: Produit Console De Migration (CDM) permettant la migration et la montée de version de Poste Asservi (PA) - OS de base: CentOS 7.4.1708				
Paquets et Vulnérabilités <input type="radio"/> Afficher/Cacher les analyses <input type="radio"/> Afficher/Cacher les détails				
Package	Liste des vulnérabilités			
	Date de création et de dernière modification	Date d'insertion	Sévérité de la vulnérabilité	Résumé de la vulnérabilité
golanggithub.com/douffes/cord v1.3.3	08/01/2024 16:45:55 / 19/01/2024 00:11:35	29/01/2024 15:05:16	UNKNOWN	CIRCL's cyber timing side-channel (yepostand)
golanggithub.com/containerd/containerd v1.7.0	19/12/2023 21:17:06 / 02/01/2024 19:11:42	29/01/2024 15:05:16	UNKNOWN	containerd allows RAPI to be accessible to a container
golanggithub.com/docker/docker v24.0.6%3Bincompatible	01/02/2024 20:51:19 / 20/03/2024 17:07:42	27/03/2024 09:07:49	MEDIUM	Classic buffer cache poisoning
golanggoogle.golang.org/grpc v1.55.0	25/10/2023 21:17:37 / 08/11/2023 04:20:52	29/01/2024 15:05:16	HIGH	gRPC-GO HTTP/2 Rapid Reset vulnerability
	10/10/2023 21:28:24 / 15/03/2024 05:21:13	29/01/2024 15:05:16	MEDIUM	HTTP/2 Stream Cancellation Attack
golanggoogle.golang.org/protobuf v1.30.0	05/03/2024 20:24:05 / 13/03/2024 21:26:31	13/03/2024 08:56:16	UNKNOWN	Intense loop in JSON unmarshaling in google.golang.org/protobuf
pygitconfigpy v4.7.2	03/04/2023 08:30:19 / 18/02/2024 05:29:13	29/01/2024 15:05:15	LOW	configgit ReadDir: exploitable by developer using values in a server-side configuration file
golanggolang.org/x/crypto v0.13.0	18/12/2023 19:22:09 / 14/03/2024 22:02:47	29/01/2024 15:05:16	MEDIUM	Prefix Truncation Attack against ChaCha20-Poly1305 and Encrypt-then-MAC also Tls1.3

Figure 20 : Exemple de rapport généré en Version 1

3.2.2.2 Résolution de bug de scan

Un autre problème a été rencontré avec l'application lors de l'insertion d'un fichier SBOM dont le nom contenait des points. Le processus de scan échouait parce que la vérification de l'extension du fichier n'était pas effectuée correctement, ce qui entraînait le rejet du fichier par l'application. Il m'a donc fallu reprendre la partie du code concernée et modifier la vérification de l'extension pour qu'elle se base sur la bonne portion du nom de fichier et ne soit pas perturbée par la présence de points supplémentaires dans le nom du fichier SBOM.

3.2.2.3 Synthèse

Ces interventions font partie des tâches courantes de maintenance et d'amélioration des applications. Elles montrent l'importance d'une écoute active des retours utilisateurs et la réactivité nécessaire pour maintenir la qualité et l'efficacité de l'application. Le processus a également mis en lumière le rôle crucial des bonnes pratiques de versionning et de gestion de code pour faciliter le suivi des changements et leur déploiement.

3.2.3 Re factoring et passage à Django

Comme mentionné précédemment, l'objectif suivant est de refactoriser l'API Flask pour la migrer vers le framework Django. Cela nécessite de revoir entièrement la conception de l'ancienne API tout en me familiarisant avec le fonctionnement de Django, afin de pouvoir transformer efficacement l'API Flask en API Django. Ce travail s'est avéré assez long car Django est un framework complexe et très élaboré.

Après avoir consacré beaucoup de temps à cette tâche, à réaliser de nombreux tests et recherches sur les technologies appropriées, j'ai décidé de procéder à la refonte de la base de données. Voici le schéma conceptuel de la base de données entièrement révisé :

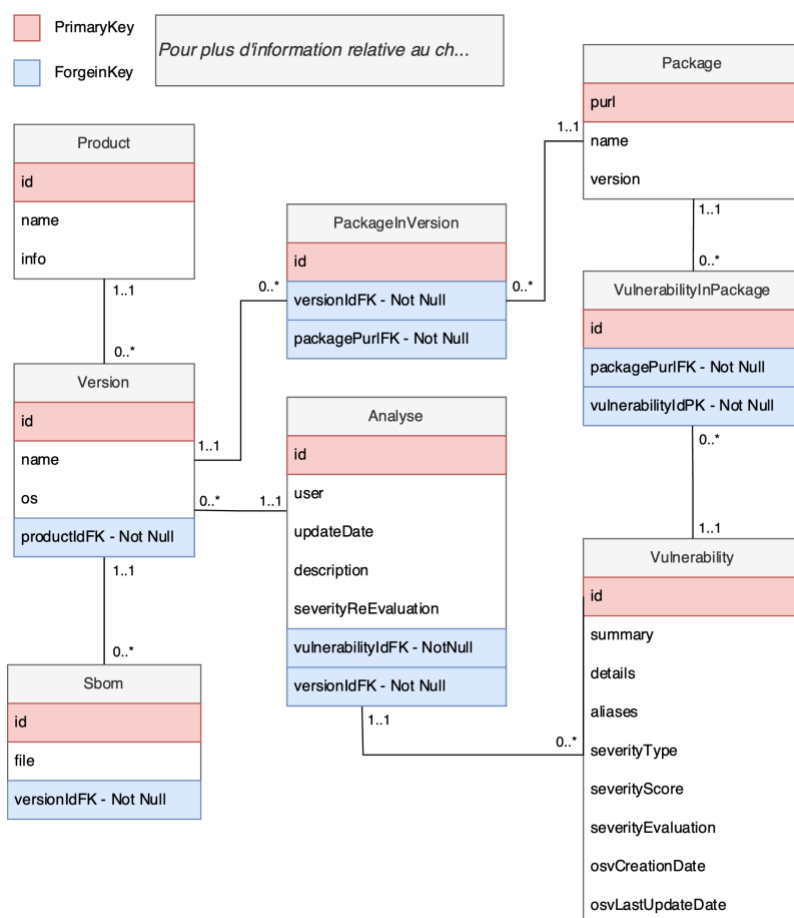


Figure 21 : Diagramme conceptuel de la BDD en Version 2

Par la suite, j'ai commencé à écrire les modèles Django qui correspondent à cette base de données. Django fonctionne avec un ORM (Object-Relational Mapping). Un ORM est un outil qui permet de convertir les données entre les systèmes de bases de données et les objets en programmation orientée objet. En d'autres termes, il s'agit d'une couche d'abstraction qui facilite la gestion des données sans avoir à écrire de requêtes SQL complexes. C'est pourquoi je passe par la rédaction de modèles Django soit des classes Python pour décrire la base de données. Par la suite, je reprendrai en main les images Docker ainsi que le Docker Compose et l'arborescence de mon projet pour l'adapter à Django.

J'ai d'ailleurs déjà construit la tram de la documentation de la futur V2 de l'application. Ci-dessous un extrait de l'arborescence de celle-ci :

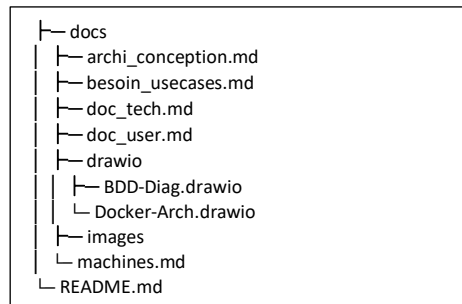


Figure 22 : Extrait de l'arborescence en Version 2

3.3 Les résultats

Au terme de cette phase d'industrialisation et de refactoring initial, nous avons significativement renforcé l'efficacité opérationnelle de l'application, tout en mettant en œuvre des bases pour les futures mises à jour. L'industrialisation via Docker a notamment permis de standardiser et de sécuriser l'environnement de déploiement, facilitant ainsi la maintenance et la mise à jour de l'application. Cette standardisation inclut également des améliorations sur l'interface utilisateur et la correction de bugs.

Le passage à Django est maintenant la priorité du projet. Ce refactoring est complexe, demandant une analyse minutieuse pour identifier les meilleurs outils et intégrer efficacement le nouveau framework. L'engagement dans cette transition nécessite une poursuite des recherches et des expérimentations pour surmonter les défis techniques et assurer une architecture robuste et performante.

La documentation du projet, que j'ai commencé à élaborer, joue un rôle crucial en facilitant l'intégration des nouveaux développeurs et en garantissant une maintenance aisée. À ce jour, le projet représente un total de 170 000 lignes de code, avec 360 heures investies dans le développement et le refactoring, ce qui témoigne de l'ampleur de l'effort fourni.

Bien que la migration vers Django ne soit pas encore achevée et qu'il reste encore beaucoup de travail à accomplir, les avancées réalisées offrent une vision encourageante pour la suite du projet. Mon engagement reste ferme dans le développement d'une application cohérente et performante, prête à s'adapter au futur produit et contrainte posé par les clients

4 Autre Réalisation


4.1 DREAM - TEST

Le programme 'DREAM' (Décodeur Réseau Enregistreur et Analyseur Multi-protocoles) est conçu pour analyser les trames TCP circulant sur les réseaux ARTERE et RMS. Opérant sous le système d'exploitation Linux CentOS, il décode en temps réel les couches IP et TCP et, selon la configuration choisie, il peut également décoder les couches ASA, ARTERE, Téléconduite, ainsi que la couche IEC 60870-5-104 et le protocole TASE.2 (qui sont spécifiques aux équipements de RTE, mais qui ne seront pas détaillés ici). L'utilisateur peut sélectionner le type de décodage à visualiser et sauvegarder ces vues dans des fichiers sur le disque. DREAM enregistre l'intégralité du trafic réseau brut pour anticiper les incidents réseau et produit des statistiques horaires sur le trafic ARTERE pour en suivre l'évolution.

Ainsi j'ai été affecté pour une période de deux jours au projet DREAM, où j'étais chargé d'exécuter des tests à partir de fiches prédéfinies, étape essentielle de la recette du produit précédant la livraison. Mon référent pour ce travail, Benjamin THOU, a été une aide précieuse, m'assistant particulièrement lors de tests complexes nécessitant des connaissances sur des produits spécifiques à RTE et l'utilisation de protocoles dédiés. Certaines tâches étaient nouvelles pour moi, et sans son aide, il m'aurait été difficile de vérifier adéquatement le fonctionnement de DREAM.

Mon rôle a donc été de compléter les fiches de test, de rapporter tous les incidents survenus lors des tests et de confirmer la réussite de ceux qui se sont déroulés sans incident.

Si dessous un exemple de fiche de test que j'ai réalisé.



Fiche de Tests

Modifiée le : 22/01/2024 11:01:00
Imprimée le : 22/01/2024 11:01:00

Fiche : T23014.docx

INDICE 5
PAGE 1/2

Création de la fiche

Date: 20/07/20

Acteur (Initiales): BT

Projet: Dream

Version: 3.1.0

Utilisation de la fiche pour validation. Attention, l'acteur doit être différent de celui qui a créé la fiche. La validation doit se faire de manière globale (sans découpage par parties)

Date: 13/02/2024

Acteur (Initiales): EF

Projet: Dream

Version: 3.1.0

Modules : TASE.2

Temps de passage de la fiche : 10 min

A intégrer à la non-régression (Oui/Non): Non

Description de la modification

Modification de la fonction decode_get_name_list_response pour vérifier la taille de l'élément à écrire avant d'effectuer l'écriture

Modification de la taille max du buffer manipulé dans la fonction ecrire_fichiers_TASE2


Architecture du test

Une machine DREAM 3

Description du test - Le test doit être déroulé dans sa globalité et non par parties.

Rejet du fichier permettant de reproduire l'anomalie

Vérification de la non-reproduction du plantage de dream



Fiche de Tests

Modifiée le : 22/01/2024 11:01:00
Imprimée le : 22/01/2024 11:01:00

Fiche : T23014.docx

INDICE 5
PAGE 2/2

Identifiant de la plateforme utilisée: IGAR/PER OLS NON-AD

(La liste des plateformes est disponible dans la GED à l'emplacement INEO/Architectures plateformes)

Procédure de test (Cocher les étapes au stylo lors du test)

✓ Sur une machine DREAM 3.

✓ Copier le fichier eth1_ET1258 disponible sous [redacted] dans le répertoire /tmp/ de la machine DREAM

✓ Arrêter toutes les instances de dream en cours d'exécution

✓ Lancer le rejeu de ce fichier à l'aide des commandes :

```
cd /tmp/dream  
./dream -i /tmp/eth1_ET1258
```

✗ Au bout de quelques secondes, constater que l'exécution de l'application se termine correctement avec un message de la forme :
[...]
2317 trames traitées sur 2317
[...]

✗ Constater l'absence de traces de la forme :
[...][Erreur de segmentation (core dumped)]

Nous avons observé un caractère aléatoire dans l'exécution de cette fiche
Il semble que DREAM produise parfois un core dump avec d'autres paquets malformés plus loin dans le fichier pcap
Ce problème de robustesse est connu et indiqué dans le dossier de livraison (§1.4 Problèmes connus)

Figure 23 : Exemple de fiche de test sur le produit DREAM

Nous retrouvons alors les annotations relatives au passage des tests, qui sont soit marquées d'une croix pour indiquer la présence d'un problème, soit annotées d'une coche pour signaler que tout s'est bien passé. Puis, à la fin du document, des informations complémentaires sur l'incident rencontré lors du passage de la fiche sont fournies.

20/04/2024

33

5 Réflexion sur le rôle de l'IA dans l'apprentissage

5.1 Contexte et contraintes professionnelles

Lorsqu'on explore le domaine de l'intelligence artificielle (IA), il est essentiel de distinguer les différents types d'IA. Une grande différence existe, par exemple, entre une IA conçue pour déterminer les cycles de lavage d'une machine à laver et des modèles de langage avancés comme Microsoft Copilot ou ChatGPT. Nous nous intéressons ici aux Modèles de Langage à Grande Échelle (LLM), qui sont des IA génératives entraînées sur de vastes quantités de données publiques et, souvent, sur les informations fournies par les utilisateurs lors de leurs interactions. La prudence est de mise lorsqu'on utilise ces outils, surtout quand ils ont accès à notre code ou lorsqu'ils traitent des données fournies dans le cadre de discussions. Comme discuté avec Guillaume PIESSAT, notre responsable qualité, il est crucial d'éviter le copier-coller de code sous licence privée dans ces plateformes pour prévenir toute violation de droits d'auteur ou de propriété intellectuelle.

Il est également important de souligner, comme nous l'avons évoqué avec Guillaume lors de nos discussions, que l'utilisation du code généré par ces modèles peut s'avérer dangereuse. En effet, les modèles peuvent avoir intégré du code issu de sources privées qui ne devraient pas être divulguées ou utilisées par d'autres entités. Nous devons donc faire preuve de prudence en gérant les informations dans les deux sens : non seulement dans ce que nous fournissons aux IA mais aussi dans l'utilisation du code qu'elles génèrent. Cela souligne la nécessité d'une utilisation réfléchie de ces technologies pour éviter la compromission de codes sous licences privées.

On pourrait alors se demander s'il existe des stratégies pour se prémunir contre ce type de risques. La réponse est oui, mais ces solutions sont souvent accessibles moyennant un coût. Nombre de ces outils, y compris ChatGPT et Copilot, proposent des offres payantes assurant la protection des données pour les entreprises et les individus qui le souhaitent. Cependant, l'allocation de budgets pour ces solutions semble encore loin d'être une priorité, que ce soit chez INEO ou dans d'autres entreprises du secteur. Une alternative consisterait à déployer localement des modèles moins performants mais entièrement sous notre contrôle et non connectés à Internet. Toutefois, cette option peut être chronophage et parfois complexe à mettre en œuvre. Pour l'instant, la solution la plus viable semble être d'utiliser ces outils de manière judicieuse et réfléchie.

5.2 Impact

En 2024, il est crucial de réfléchir à l'impact que l'intelligence artificielle (IA) peut avoir sur l'apprentissage, notamment dans le contexte de l'alternance. Mon expérience chez INEO illustre bien comment l'IA peut transformer le parcours d'apprentissage.

Prenons l'exemple de l'apprentissage d'une technologie quelconque qui débute au niveau 0, avec pour objectif d'atteindre le niveau maximal, soit le niveau 10. Avec l'arrivée de l'IA, un alternant débutant au niveau 3 ou 4 pourrait, grâce à ces outils, réaliser des tâches de niveau 7 ou 8, même sans en maîtriser tous les détails. Ces technologies permettent donc une approche d'apprentissage non linéaire, comme si ont commencé la construction d'une maison par les murs sans en avoir terminé les fondations, elle en sera plus fragile au départ mais au fil du temps et des travaux elle pourra être consolidée.

Par exemple, lors de mon expérience chez INEO, j'ai utilisé le framework Django sans avoir préalablement une expertise en Python. Grâce à l'assistance de ChatGPT, qui m'a fourni des exemples et des explications, j'ai pu débiter le développement de mon application Django. Bien que je ne maîtrisais pas tous les détails, cette aide m'a permis de réaliser des tâches plus complexes que si j'avais dû me limiter à la lecture de la documentation seule.

Cette capacité à opérer avec plus d'autonomie a été remarquée et valorisée par mon référent Yannick, comme souligné lors de l'entretien avec mon tuteur pédagogique, Jean-Michel BRUEL. Cette expérience démontre que l'IA peut effectivement augmenter l'efficacité de l'apprentissage, en permettant aux apprentis de réaliser des tâches plus complexes plus tôt dans leur formation, tout en développant progressivement une compréhension approfondie.

5.3 Conclusion

L'impact de l'intelligence artificielle (IA) sur l'apprentissage est profondément transformateur, comme le démontre clairement mon expérience chez INEO. En permettant aux apprentis de débiter leur formation à un niveau supérieur, l'IA accélère effectivement le processus d'apprentissage, permettant ainsi la réalisation de tâches plus complexes dès le début. Toutefois, cette avancée ne vient pas sans ses défis. En commençant par des niveaux plus élevés sans une compréhension complète des bases, les fondations de l'apprentissage peuvent initialement se révéler moins solides.

Cela soulève une préoccupation importante : bien que la capacité à effectuer des tâches complexes soit accrue, il est essentiel de ne pas négliger les fondamentaux qui soutiennent la compréhension globale et durable dans un domaine. L'adoption de l'IA doit donc être équilibrée avec une approche pédagogique qui assure un renforcement continu des bases tout en exploitant les capacités avancées offertes par ces technologies.

L'objectif devrait être de construire un parcours d'apprentissage qui, tout en intégrant les avantages de l'IA, permet également un développement robuste des compétences fondamentales. Cela garantit que les apprentis ne sont pas seulement capables d'exécuter des tâches, mais qu'ils possèdent aussi une compréhension profonde des principes sous-jacents nécessaires pour innover et s'adapter à des environnements en constante évolution.

En résumé, si l'IA transforme le paysage éducatif en permettant des progrès rapides, elle doit être utilisée judicieusement pour renforcer, plutôt que de compromettre, les bases solides de l'apprentissage et du développement professionnel.

6 Bilan

6.1 Les objectifs et leur impact

Aujourd'hui, j'ai donc atteint ou commencé à atteindre les objectifs qui m'ont été assignés, en commençant par l'industrialisation de l'application. Actuellement, la version 1 de l'application est pleinement fonctionnelle et disponible au sein du réseau d'INEO. Elle répond au besoin primaire de générer facilement des rapports de vulnérabilité pour nos clients.

Mon second objectif a été de pouvoir répondre aux problématiques rencontrées sur l'application et de les résoudre, ce que j'ai réussi à faire à chaque fois que cela m'a été demandé.

Aujourd'hui, INEO dispose donc d'une version fonctionnelle entre les mains ; cependant, cette version n'est pas pérenne, est mal documentée et difficilement maintenable. C'est pourquoi je travaille assidûment au développement de la version 2, qui sera plus modulaire et plus facile à maintenir.

6.2 La gestion de projet

En ce qui concerne la gestion de projet, je suis maintenant capable de me débrouiller presque entièrement seul, ayant une compréhension approfondie du fonctionnement de l'entreprise, de la DSI aux gestionnaires des dépôts ou du proxy qui permet de mettre à disposition des ressources sur le réseau d'INEO. J'ai également acquis une expertise dans la coordination des différentes phases du projet, de la planification initiale à la mise en œuvre, et je peux désormais gérer les urgences et les imprévus avec assurance et efficacité.

6.3 Apports personnels

Le travail que j'ai eu l'opportunité de réaliser chez INEO m'a non seulement permis de découvrir le monde professionnel mais aussi de solidifier et d'étendre mes compétences techniques, notamment ma maîtrise de Docker. Cette expertise acquise en environnement professionnel me permet désormais de contribuer à des répertoires publics sur des modules Python pour étendre les capacités de Django, renforçant ainsi mon profil dans la communauté open source. Ces contributions m'ouvrent des perspectives d'avenir passionnantes, tant en termes de carrière dans le développement de logiciels que de participation à des projets innovants à plus grande échelle.

Conclusion

Le projet Gestion Faille a pour objectif principal de garantir la sécurité des produits fournis aux clients. Grâce à l'industrialisation et à la maintenance rigoureuse que j'ai effectuées sur la version un de l'application Gestion Faille, celle-ci répond désormais efficacement à ce besoin essentiel. Cela contribue à renforcer la confiance des clients dans la fiabilité et la sûreté des solutions offertes par INEO.

Cette alternance a été une expérience profondément enrichissante. Elle m'a permis de mettre en pratique mes connaissances théoriques tout en développant des compétences pratiques cruciales dans un contexte professionnel réel. J'ai eu l'opportunité d'explorer et de comprendre le fonctionnement interne d'une entreprise technologique, et de contribuer significativement à des projets essentiels. Grâce à ces expériences, j'ai pu constater une évolution notable de mes compétences et de ma capacité à m'adapter et à collaborer efficacement au sein d'une équipe.

L'alternance m'a également offert l'occasion de comprendre l'importance de la sécurité informatique dans le contexte commercial actuel et de participer activement à l'amélioration des processus de sécurité chez INEO. La version 2 de l'application Gestion Faille, sur laquelle je travaille actuellement, introduira une série de fonctionnalités supplémentaires qui simplifieront davantage la tâche. Ces innovations visent à automatiser et à optimiser le processus d'analyse des failles, rendant ainsi le suivi des vulnérabilités encore plus efficace.

En résumé, cette période a non seulement élargi mon horizon professionnel, mais a également renforcé mon engagement envers une carrière dans le développement de solutions technologiques sécurisées.

Lexique

- **SBOM :**

Un SBOM, ou "Software Bill of Materials" (facture des matériaux logiciels), est une liste exhaustive de tous les composants, notamment les bibliothèques, les modules, les packages et les snippets de code, qui sont utilisés pour construire un logiciel ou un OS. Il inclut des informations sur les versions, les licences, et parfois les sources des composants. Le SBOM est utilisé pour améliorer la transparence, la gestion des risques et la sécurité des logiciels, en permettant aux utilisateurs et aux développeurs de comprendre exactement quels composants logiciels sont intégrés dans un produit. Cela aide également à gérer les vulnérabilités, à respecter les obligations légales et réglementaires en matière de licences, et à effectuer des audits de sécurité plus efficaces.

- **OSV :**

OSV (Open Source Vulnerabilities) est une base de données de vulnérabilités qui concerne les projets open source. Elle vise à fournir une source fiable et automatisée de données sur les vulnérabilités dans les logiciels open source, en utilisant un système qui facilite la remontée, la mise à jour et l'interrogation des informations sur les vulnérabilités. OSV s'efforce de maintenir une compatibilité avec les normes existantes telles que les CVE et autres bases de données de vulnérabilités, tout en étant plus accessible et automatisé.

- **OSV Scanner :**

L'OSV Scanner est un outil qui utilise la base de données OSV pour scanner les logiciels à la recherche de vulnérabilités connues. Cet outil analyse les dépendances d'un projet pour identifier les composants qui sont affectés par des vulnérabilités répertoriées dans la base de données OSV. Il fournit ainsi une évaluation des risques potentiels et des recommandations pour la mise à jour ou le remplacement des composants vulnérables. L'OSV Scanner est particulièrement utile dans les processus de développement logiciel pour intégrer la sécurité dès la conception et maintenir la sûreté des applications tout au long de leur cycle de vie.

- **Mix énergétique :**

Le mix énergétique fait référence à la combinaison des différentes sources d'énergie utilisées pour répondre aux besoins énergétiques d'une région, d'un pays, ou du monde entier. Ce terme englobe toutes les formes d'énergie disponibles, qu'elles soient renouvelables (comme l'énergie solaire, éolienne, hydroélectrique ou géothermique) ou non renouvelables (comme le charbon, le pétrole ou le gaz naturel). L'objectif du mix énergétique est souvent d'optimiser la balance entre la sécurité énergétique, la viabilité économique et la durabilité environnementale.

- **FullStack :**

Le terme "FullStack" fait référence à une compétence en développement informatique qui englobe à la fois le développement côté client (Front-end) et le

développement côté serveur (Back-end). Un développeur FullStack est capable de gérer tous les aspects de la construction d'une application web ou mobile, depuis l'interface utilisateur jusqu'à la base de données, en passant par la logique serveur.

- **FrontEnd :**

Le FrontEnd fait référence à la partie d'une application ou d'un site web qui est visible et directement accessible par l'utilisateur. Il englobe la conception de l'interface utilisateur (UI), l'expérience utilisateur (UX), ainsi que le développement des fonctionnalités interactives. Les technologies de FrontEnd typiques incluent HTML, CSS et JavaScript, ainsi que des frameworks comme Angular, React ou Vue.js, qui aident à structurer et à animer les interfaces de manière dynamique.

- **BackEnd :**

Le BackEnd désigne la partie serveur d'une application web ou d'un service en ligne, où s'opèrent la logique métier, le traitement des données et les interactions avec la base de données. Cette couche est responsable de la gestion des requêtes envoyées par le FrontEnd, l'exécution des opérations nécessaires, et l'envoi des réponses adéquates. Les technologies de BackEnd comprennent des langages de programmation comme Python, Java ou Node.js, et des frameworks associés tels que Django pour Python ou Spring pour Java.

- **LocalHost :**

LocalHost est un terme informatique qui fait référence à l'ordinateur sur lequel un programme est en train de s'exécuter. En d'autres termes, il désigne l'ordinateur "local" utilisé par une personne. Par exemple, lorsque des développeurs créent des sites web ou des applications, ils peuvent tester leur travail sur leur propre ordinateur en utilisant LocalHost comme une sorte d'adresse qui pointe vers leur machine. Cela leur permet de voir comment le site ou l'application fonctionne avant de le mettre en ligne sur Internet. LocalHost sert essentiellement de plateforme de test interne pour les développeurs.

- **MariaDB :**



MariaDB est un système de gestion de base de données relationnelle, forké de MySQL, qui propose une alternative open-source sous licence GPL. Il a été conçu pour être hautement compatible avec MySQL, tout en offrant de nouvelles fonctionnalités, des améliorations de performance et une meilleure ouverture vers la communauté. MariaDB est couramment utilisé pour stocker des données qui peuvent être requêtées par des applications web, et il est géré par une fondation indépendante, la MariaDB Foundation.

- **Flask :**



Flask est un micro-framework open source pour le développement web en Python. Il est conçu pour rendre le développement d'applications web simples et rapides, avec la capacité de s'étendre pour des applications plus complexes. Flask fournit les outils, les bibliothèques et les technologies nécessaires pour construire une application web, tout en permettant aux développeurs de choisir les composants les plus adaptés pour leur projet.

- **Angular :**



Angular est un framework open source de développement d'applications web développé par Google. Il permet de créer des applications web dynamiques et riches en fonctionnalités en utilisant TypeScript, qui est une surcouche de JavaScript. Angular se distingue par sa capacité à faciliter le développement de Single Page Applications (SPA), offrant une expérience utilisateur fluide et réactive grâce à la mise à jour dynamique du contenu sans avoir à recharger la page.

- **Django :**



Django est un framework de développement web en Python qui permet de construire des sites web complexes de manière rapide et avec moins de code. Il suit le modèle de conception "Model-View-Template" (MVT), ce qui le rend très efficace pour le développement d'applications web dynamiques. Django intègre des fonctionnalités qui facilitent la gestion des bases de données, le traitement des formulaires, l'authentification des utilisateurs, et bien plus encore.

- **Git:**



Git est un système de contrôle de version distribué conçu pour gérer efficacement les projets de toute taille avec rapidité et précision. Il permet aux développeurs de sauvegarder des versions de leur code à travers des "commits", qui enregistrent les changements effectués sur les fichiers à un moment donné. Les développeurs peuvent également créer des "branches" pour développer des fonctionnalités, corriger des bugs ou expérimenter en

isolant ces changements du code principal. Lorsque les modifications d'une branche sont prêtes et testées, elles peuvent être "mergées", c'est-à-dire fusionnées, dans la branche principale ou dans d'autres branches du projet, intégrant ainsi de manière cohérente les développements parallèles.

- **GitLab:**



GitLab est une plateforme de gestion de dépôts Git qui fournit une interface graphique, des outils de gestion de cycle de vie du développement logiciel, ainsi que des fonctionnalités d'intégration continue. Parmi ses nombreuses fonctionnalités, GitLab permet de gérer les "merge requests", qui sont des propositions de fusion de code d'une branche vers une autre. Une merge request permet non seulement de proposer des changements, mais aussi de les discuter, de les réviser et de les tester avant de les intégrer définitivement dans la branche principale du projet. GitLab offre également la gestion des "issues", qui sont utilisées pour suivre les tâches, les améliorations et les bugs au sein d'un projet. Les issues permettent une organisation claire du travail à faire et facilitent la communication et la collaboration entre les membres de l'équipe sur des points spécifiques du projet, rendant ainsi la gestion de projet plus efficace et transparente.

- **Copilot :**



GitHub Copilot est un outil de développement assisté par intelligence artificielle, conçu pour aider les programmeurs en suggérant automatiquement des lignes de code et des fonctions en temps réel. Il est basé sur une série de modèles de langage formés sur des milliards de lignes de code public, ce qui lui permet de comprendre le contexte du code écrit et de proposer des complétions pertinentes.

- **ChatGPT :**



ChatGPT est un modèle de langage développé par OpenAI basé sur l'architecture GPT (Generative Pre-trained Transformer). Il est conçu pour générer du texte qui suit le style et le contexte des prompts fournis par les utilisateurs. ChatGPT peut être utilisé pour une variété d'applications, y compris la réponse à des questions, la création de contenu écrit, la traduction de langues, et plus encore, en imitant une conversation humaine de manière cohérente et contextuellement appropriée.

- **Environnement virtuel Python ou autre :**

Un environnement virtuel en Python, ou dans d'autres langages de programmation, est un espace isolé qui permet d'exécuter des programmes tout en ayant leurs dépendances spécifiques gérées séparément des autres projets. Cela permet aux développeurs d'éviter les conflits entre les versions des bibliothèques et de faciliter la gestion des dépendances pour chaque projet.

En Python, un environnement virtuel peut être créé à l'aide de modules comme `venv` ou `virtualenv`. Une fois activé, cet environnement utilise sa propre version de l'interpréteur Python et ses propres bibliothèques installées, indépendamment des bibliothèques installées globalement sur le système. Cela permet aux développeurs de travailler sur plusieurs projets avec différentes exigences de dépendances sans interférence, améliorant ainsi la reproductibilité et la maintenance des applications.

Table des illustrations

Figure 1 : Frise chronologique de l'historique d'INEO	6
Figure 2 : Organigramme du groupe Bouygues	6
Figure 3 : Organigramme du groupe EQUANS	7
Figure 4 : Organigramme simplifié de l'agence INEO MPLR	7
Figure 5 : Liste non exhaustive des clients d'INEO	8
Figure 6 : Exemple d'interface de visualisation des données du projet Tahiti	9
Figure 7 : Représentation de l'application réalisé par Noe Fauche	13
Figure 8 : Diagramme de cas d'utilisation de l'application Gestion Faille	14
Figure 9 : Schéma de la base de données OSV	15
Figure 10: Exemple de merge request sur GitLab	17
Figure 11 : Gantt réalisé	18
Figure 12 : DockerFile du BackEnd en version V1	20
Figure 13 : DockerFile de la base de données	22
Figure 14 : DockerFile du FrontEnd en Version 1	23
Figure 15 : Arborescence du projet en Version 1	25
Figure 16 : Schéma de base de données en Version 1	26
Figure 17 : Schéma représentatif de l'app en Version 1	26
Figure 18 : Illustration du problème de "vv" dans le rapport	28
Figure 19 : Colonnes renommées dans le rapport	29
Figure 20 : Exemple de rapport généré en Version 1	29
Figure 21 : Diagramme conceptuel de la BDD en Version 2	31
Figure 22 : Extrait de l'arborescence en Version 2	32
Figure 23 : Exemple de fiche de test sur le produit DREAM	33
Figure 24 : Diagram de fonctionnement de Docker	45
Figure 25 : Exemple de DockerFile	46

Annexes

Introduction à Docker, Dockerfile, et Docker Compose

Docker est une plateforme de conteneurisation qui permet de "packager" une application et ses dépendances dans un conteneur virtuel qui peut s'exécuter sur n'importe quel système d'exploitation Linux, Windows ou macOS supportant Docker. Cette technologie assure la portabilité, la cohérence des environnements d'exécution, et facilite ainsi le développement, les tests, et la production en minimisant les différences entre les environnements.

- **Dockerfile et Build d'Images**

Un Dockerfile est un script composé de diverses instructions qui permettent de construire une image Docker. Chaque instruction dans un Dockerfile ajoute une couche à l'image, et chaque couche est stockée de manière incrémentielle par rapport à la précédente. Les instructions courantes incluent :

- FROM : définit l'image de base à utiliser.
- RUN : exécute des commandes dans le conteneur.
- COPY ou ADD : copie des fichiers ou répertoires dans le conteneur.
- CMD : définit la commande par défaut à exécuter lors du démarrage du conteneur.
- EXPOSE : indique les ports sur lesquels un conteneur écoute les connexions.

Lorsque le Dockerfile est complété, la commande `docker build` permet de créer une image Docker qui peut être exécutée pour démarrer des conteneurs.

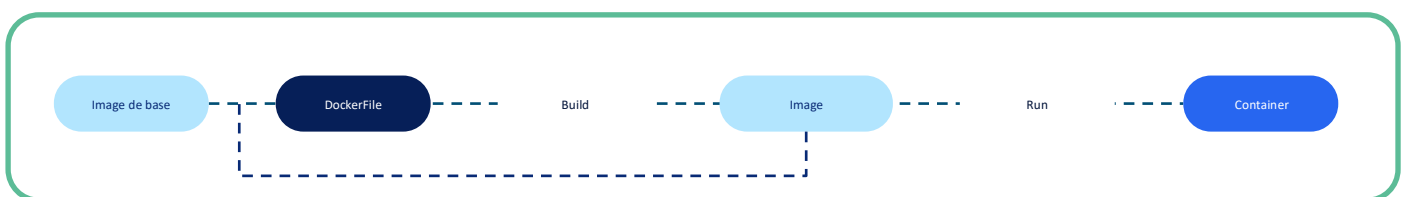


Figure 24 : Diagram de fonctionnement de Docker

- Exemple de DockerFile :

```
FROM python:3.12-slim
COPY ./requirements.txt /
RUN pip install -r /requirements.txt
RUN rm -rf /requirements.txt
WORKDIR /app
RUN python manage.py makemigrations
RUN python manage.py migrate
CMD [ "python", "manage.py", "runserver", "0.0.0.0:8000" ]
```

Figure 25 : Exemple de DockerFile

Dans cet exemple, nous avons un Dockerfile destiné à préparer un environnement pour une application web Django. Le processus commence avec une image de base "slim" de Python 3.12, fournie par les mainteneurs de l'interpréteur Python. Utiliser une image de base officielle est un moyen sûr d'avoir un environnement de départ stable et bien testé. De plus, la version "slim" est choisie parce qu'elle est allégée, ne contenant que les outils et les bibliothèques essentiels, ce qui est en accord avec une des meilleures pratiques Docker : maintenir les images aussi légères que possible pour réduire le temps de téléchargement, économiser de l'espace de stockage et accélérer le déploiement.

Le Dockerfile procède ensuite à la copie du fichier requirements.txt dans le conteneur, qui liste toutes les dépendances nécessaires pour l'application. En utilisant la commande pip, ces dépendances sont installées dans le conteneur. Après l'installation, le fichier requirements.txt est supprimé pour ne pas encombrer l'image avec des fichiers inutiles.

La création d'un espace de travail spécifique dans le conteneur est établie en définissant un répertoire de travail /app. Cette pratique est courante pour aider à organiser les fichiers et à clarifier où les commandes suivantes seront exécutées.

Avant de démarrer l'application, des étapes de préparation de la base de données sont nécessaires. Le Dockerfile exécute les commandes makemigrations et migrate de Django, qui préparent et appliquent des schémas de base de données. Cette automatisation assure que le conteneur est prêt à être utilisé dès qu'il est lancé.

Enfin, la commande par défaut pour lancer l'application est spécifiée. Elle est configurée pour démarrer le serveur de développement de Django et écouter sur le port 8000 accessible de l'extérieur du conteneur. Cela signifie

que dès que le conteneur est en cours d'exécution, l'application est prête à recevoir des requêtes.

Ce Dockerfile est un exemple typique de la façon dont Docker peut simplifier la configuration et le déploiement d'applications web tout en veillant à maintenir l'environnement aussi propre et léger que possible.

- **Docker Compose**

Docker Compose est un outil pour définir et gérer des applications multi-conteneurs. Avec un fichier YAML de configuration, Docker Compose permet de configurer les aspects suivants d'une application :

- Services : Chaque service peut être basé sur une image construite à partir d'un Dockerfile ou une image déjà existante.
- Réseaux : Docker Compose configure des réseaux internes pour faciliter la communication entre les conteneurs. Les conteneurs d'un même réseau peuvent communiquer entre eux en utilisant leurs noms de service comme hôtes.
- Volumes : Pour la persistance des données, Docker Compose permet de monter des volumes, soit sur le conteneur seul, soit entre le conteneur et le système hôte.
- Ports : Les ports sur lesquels les services sont accessibles de l'extérieur peuvent être configurés dans Docker Compose, permettant ainsi de diriger le trafic entrant vers des services spécifiques au sein des conteneurs.

- **Gestion des Builds avec Docker Compose**

En plus de la gestion de l'exécution, Docker Compose peut également gérer le processus de build des images via la définition des services dans le fichier `docker-compose.yml`. Cela inclut la construction automatique des images basées sur des Dockerfiles personnalisés avant de démarrer les conteneurs, en utilisant l'option `build` dans la définition d'un service.

En résumé, Docker et Docker Compose offrent une plateforme robuste pour la gestion du cycle de vie des applications, depuis le développement jusqu'à la production, en assurant une portabilité, une scalabilité et une cohérence environnementale optimales. Un schéma illustratif des étapes de Docker à partir du Dockerfile pourrait grandement aider à visualiser ce processus.

TRAITEMENT DOCUMENTAIRE

NOM ETUDIANT : FOURNET Enzo

Date de Naissance : 13/09/2004

Signature

NOM TUTEUR IUT : BRUEL Jean-Michel

DEPT. – ANNEE – PROMO : INFO-2023/2024-BUT3

NOM ENTREPRISE : INEO

Ville- Pays : Toulouse France

NOM TUTEUR ENTREPRISE : CASTILLO Luc

Signature

☐ **Confidentiel ***

SUJET DE STAGE : Développement d'une solution de sécurité

MOTS CLES (5) : FullStack, Docker, Sécurité, OSV, SBOM

RESUME :

Au cours de mon alternance chez INEO, j'ai été impliqué dans le développement et la refonte d'une application FullStack utilisant Angular et Flask, puis Django, avec une intégration via Docker pour le déploiement et le développement de l'application. Cette application avait pour objectif d'utiliser OSV Scanner pour détecter les vulnérabilités dans tous les paquets installés sur un produit, via un SBOM. Cette expérience m'a permis de participer activement à la sécurisation et à l'optimisation de solutions critiques, en réponse à la menace croissante des cyberattaques. Mon objectif a donc été de maintenir l'application existante, de l'améliorer et de commencer le refactoring pour créer une version 2.0 plus flexible et robuste.

NB : Votre signature valide et donne droit à la bibliothèque de l'IUT de Toulouse II Blagnac, de diffuser ce document dans le catalogue commun du réseau des bibliothèques des universités de Toulouse.

Portail du réseau : http://bibliotheques.univ-toulouse.fr/38310688/0/fiche___pagelibre/&Rt=

*** Mettre une croix si confidentiel**