

**IMT Mines Alès**6 Avenue de Clavières, 30319 Alès Cedex

Rapport de TP

Sécurité applicative –

INFRES 17

Session 2024-2027

$ ls /Equipe

HUMBERT Thomas

ANDRADE Julien

SODDU Valentine

CARLINO Romain

GALLO Louis

SUARD Erwann

$ cat Projet.md

\*\*TODO Trouver un nom pour le projet\*\*

$ date

$ getent group Professeur

Professeur:x:1001:Mr\_MEGE

Mon Jun 23 18:00:00 CEST 2025

Professeur :

Sommaire

[i- Présentation 3](#_Toc201586568)

[1 – Introduction 3](#_Toc201586569)

[2 – Contexte 3](#_Toc201586570)

[ii- ANdrade julien 4](#_Toc201586571)

[iii- carlino romain 7](#_Toc201586572)

[iv- gallo louis & humbert thomas 10](#_Toc201586573)

[v- soddu valentine 13](#_Toc201586574)

[vi- suard erwann 17](#_Toc201586575)

[vii- améliorations 21](#_Toc201586576)

[viii- conclusion 23](#_Toc201586577)

# i- Présentation

**1. Introduction**

Avec la montée en puissance des architectures cloud et des applications distribuées, Kubernetes est devenu indispensable pour le déploiement, la gestion et la scalabilité des applications. Ce travail pratique nous a permis de mettre en œuvre un déploiement sécurisé, en appliquant les principes fondamentaux de sécurité et de fiabilité vus en cours.

Nous avons ainsi conçu une infrastructure basée sur trois applications interconnectées au sein d’un cluster Kubernetes :

* Un **client** : application en ligne de commande permettant d’interagir avec le serveur.
* Un **serveur** : une API REST assurant la gestion des requêtes et l’accès aux données.
* Une **interface web (IHM)** : facilitant l’interaction avec le serveur via une interface utilisateur.

Tout au long du TP, nous avons suivi les principes **ICnRA-FDMS**, garantissant un déploiement sécurisé et fiable.

**2. Contexte**

Dans le cadre de la croissance de notre entreprise de vente de jus de fruits en ligne, nous avons engagé une migration vers Kubernetes afin de moderniser notre infrastructure et d’améliorer la fiabilité, la sécurité et la scalabilité de nos services.

# ii- ANdrade julien

**Responsable : Base de Données (BDD) et Rapport**

Julien ANDRADE, notre premier technicien (par ordre alphabétique), a été en charge de la **conception, de la sécurisation et de la mise en place de la base de données** utilisée dans notre plateforme de vente de jus de fruit en ligne.

**Travaux réalisés :**

* **Schéma conceptuel et physique de la base de données** :

Julien a conçu l’organisation des différentes entités (clients, transactions, jus, ingrédients, etc.) et a réfléchi aux relations entre ces données pour assurer une **cohérence** et une **intégrité** optimale.

* **Écriture du code SQL** :

Il a réalisé la structure complète des tables, avec une attention particulière aux contraintes d'intégrité référentielles (clés étrangères, suppressions en cascade, contrôles de quantités, unicité des emails, etc.).

* **Mise en place de la sécurité des mots de passe** :

➔ Les mots de passe ne sont pas stockés dans la table des clients directement.  
➔ Une **table dédiée Client\_Password** a été créée pour :

* + Séparer les données sensibles des informations publiques des clients.
  + Permettre une gestion plus fine des mises à jour de mot de passe.
  + Favoriser de futures améliorations de sécurité (ex : gestion de l'historique des mots de passe, politique de renouvellement, etc.).
  + Les mots de passe sont **stockés sous forme de hash** et jamais en clair, ce qui respecte les bonnes pratiques de sécurité.
* **Configuration Docker sécurisée (YAML)** :

Julien a préparé le fichier docker-compose.yml pour déployer la base de données MySQL avec :

* + Un **mot de passe root fort et configurable** via la variable d’environnement MYSQL\_ROOT\_PASSWORD.
  + Un **fichier SQL d’initialisation** intégré de manière contrôlée.
  + Un **healthcheck** automatique pour vérifier que la base de données est correctement lancée et opérationnelle.
* Mi**se à jour dynamique de la BDD** :

Julien a régulièrement **adapté la structure de la base de données en fonction des décisions collectives de l'équipe**, assurant ainsi la cohérence avec les besoins fonctionnels et techniques en évolution.

* **Élaboration du rapport final** :

Il a mené des **entretiens avec chaque membre de l’équipe** afin de :

Centraliser les contributions individuelles.

Mettre en avant les choix sécuritaires réalisés tout au long du projet.

Présenter de manière claire et structurée le travail de chacun.

**Informations complémentaires sur la base de données :**

* Les relations entre clients, transactions et jus ont été pensées pour permettre une **traçabilité complète** des commandes.
* Les suppressions en cascade assurent qu'aucune donnée orpheline ne persiste (ex : si un client est supprimé, ses transactions et ses mots de passe le sont aussi).
* Les quantités de jus et d’ingrédients sont **contrôlées par des contraintes SQL** pour éviter les erreurs de saisie.

Une image contenant texte, capture d’écran, diagramme, Police

Le contenu généré par l’IA peut être incorrect.*Schéma de la base de données*

# iii- carlino romain

**Responsable : Backend, liaison IHM, Client ligne de commande et Base de Données**

Romain a pris en charge le **développement du backend** en utilisant **Python avec le framework FastAPI**, permettant de faire le lien entre :

* L’interface web (IHM)
* Le client en ligne de commande
* La base de données

**Travaux réalisés :**

**Mise en place du backend avec FastAPI**

* Choix du framework **FastAPI**, reconnu pour sa rapidité et son support natif d’OpenAPI (Swagger).
* Création d'une **API performante, bien documentée et facilement extensible.**
* Mise en place d’une **documentation sécurisée et interactive** via OpenAPI pour faciliter les tests tout en contrôlant les accès.

**Structuration et validation des données avec Pydantic**

* Utilisation des **Pydantic models** pour définir précisément les structures de données :
  + **Validation automatique des types, des tailles minimales et maximales, et des formats attendus.**
  + **Génération d’erreurs automatiques** si les données reçues ne respectent pas le modèle, bloquant les traitements malveillants dès la réception.
  + **Sérialisation et désérialisation JSON facilitées** et sécurisées.
* Cette approche garantit que **les données échangées entre les services respectent toujours un format fiable et contrôlé.**

**Intégration de design patterns**

* Mise en place de **design patterns adaptés** (tels que le pattern service ou repository) pour :
  + **Séparer les responsabilités** dans le code.
  + Faciliter la **maintenance et l’évolutivité** du backend.
  + Renforcer la **clarté et la robustesse** de l’application.

**Mise en place de pratiques de sécurité essentielles**

* **Stockage sécurisé des secrets** :
  + Utilisation des **variables d’environnement** pour gérer les mots de passe et les clés sensibles.
  + **Aucune donnée critique stockée en clair dans le code source.**
* **Gestion sécurisée des erreurs** :
  + Les retours sont volontairement **génériques et neutres** (ex. : "Erreur lors du traitement de la demande") afin de **ne jamais exposer la structure interne ou les failles potentielles**.
* **Approche défensive systématique** :
  + En cas de données incohérentes ou ambiguës, l’application **interrompt immédiatement le traitement** plutôt que de continuer de manière instable.
  + Cette philosophie prévient la propagation d’erreurs et limite les vecteurs d’attaque.

**Choix stratégiques pour limiter la surface d’attaque**

* Romain a **refusé d’intégrer des fonctionnalités jugées trop exposées**, comme un profil "magasinier" permettant la gestion des stocks via l’IHM.
* Les stocks sont gérés uniquement via **modification directe dans la base de données par des administrateurs** disposant d’un accès sécurisé.

**Difficultés rencontrées**

* Configuration sécurisée de l’environnement :
  + Création et optimisation des **fichiers Dockerfile**.
  + Mise en place de **certificats TLS** pour chiffrer les échanges entre les services.
* Gestion des droits et sécurisation des routes exposées via FastAPI.

*Une image contenant texte, capture d’écran, nombre, Police

Le contenu généré par l’IA peut être incorrect.*

*Documentation OpenAPI*

# iv- gallo louis & humbert thomas

**Responsables : Infrastructure Kubernetes, persistance des données, sécurisation des communications entre services**

Louis et Thomas ont travaillé ensemble sur la **mise en place et la sécurisation de l’environnement Kubernetes**, garantissant la disponibilité, la sécurité et la communication efficace entre nos différents services.

**Travaux réalisés :**

**Orchestration et redondance des services via Kubernetes**

* **Déploiement de plusieurs pods pour chaque service** grâce aux *deployments* Kubernetes. Cette redondance assure une haute disponibilité des services : si un pod tombe, les autres continuent à gérer la charge, et Kubernetes relance automatiquement le pod défaillant.
* **La base de données est déployée avec un volume persistant** (persistent volume claim), garantissant que les données restent conservées malgré la nature éphémère des pods. Ce volume est limité à 1 Go, équilibrant stockage et performances.

**Sécurisation des communications internes**

* Mise en place des *services Kubernetes* associés à *Istio* pour gérer la communication entre les pods via un réseau interne sécurisé.
* **Implémentation de *NetworkPolicies***, une évolution importante par rapport au projet initial : ces règles limitent la communication réseau aux pods autorisés, notamment en restreignant l’accès à l’API pour que seule l’IHM puisse la contacter. Cela renforce la sécurité en empêchant toute communication non désirée ou potentiellement malveillante.
* Cette politique réduit considérablement la surface d’attaque et empêche des communications non autorisées dans l’environnement interne.

**Accès externe sécurisé**

* Déploiement d’une *gateway* Kubernetes et d’une ressource *HTTPRoute* pour gérer les accès externes aux services autorisés, essentiellement l’IHM et le client.
* Mise en place de la communication **HTTPS via certificats Let's Encrypt**. Même si ces certificats sont adaptés pour des environnements de test ou éducatifs, ils garantissent la confidentialité et l’intégrité des échanges.
* Il n’est **pas conseillé d’ouvrir une API sur le web** car cela représente une **vulnérabilité supplémentaire** que des attaquants peuvent exploiter. Lors de nos tests, nous avons dû exceptionnellement exposer l’API pour résoudre un problème technique, mais ce choix reste risqué et doit être évité autant que possible en production.

Depuis l’extérieur :

* Seuls l’IHM et le client sont autorisés à accéder aux services via la gateway.
* Tous les autres services sont isolés et inaccessibles depuis l’extérieur.

**Choix techniques et sécuritaires :**

* Initialement, **aucune politique réseau n’était prévue**. Mais les exigences du cahier des charges ont conduit à adopter les *NetworkPolicies* afin de **réduire la surface d’attaque** en limitant l’accès à l’API uniquement depuis l’IHM.
* La persistance des données via volumes assure la **continuité de service et l’intégrité des informations** en cas de redémarrage des pods.
* La redondance des pods offre une **résilience accrue**, garantissant que le système reste disponible même en cas de défaillance d’un composant.
* Le protocole HTTPS renforce la **confidentialité** des échanges et protège les données contre les interceptions.
* **Prudence sur l’exposition web des API** : éviter d’ouvrir une API au public afin de limiter les risques d’exploitation par des hackers.

**Pour plus de détails (Réalisés par Thomas et Louis) :**

[**https://github.com/RACunderskrt/secuDocs/blob/main/SECU/SecuApp.md**](https://github.com/RACunderskrt/secuDocs/blob/main/SECU/SecuApp.md)

**Synthèse sécurité applicative Kubernetes :**

|  |  |  |
| --- | --- | --- |
| Aspect | Implémentation | Bénéfices sécurité |
| Redondance des pods | Deployment Kubernetes | Haute disponibilité |
| Volumes persistants | Persistent Volume Claim | Intégrité et conservation des données |
| Communication interne | Services + Istio | Routage sécurisé |
| Restriction d’accès | NetworkPolicies | Limitation des accès non autorisés |
| Accès externe sécurisé | Gateway + HTTPRoute + HTTPS (Let’s Encrypt) | Confidentialité et intégrité |

**Difficultés rencontrées :**

* La configuration des *NetworkPolicies* a été un défi majeur, nécessitant une compréhension fine du réseau Kubernetes et une adaptation pour répondre aux besoins de sécurité sans nuire à la communication interne.
* La gestion des volumes persistants a demandé une attention particulière pour garantir la cohérence et éviter les pertes de données en cas de redémarrage.

# v- soddu valentine

**6 – Rôle et contributions de Valentine**

**Responsable : Interface Homme-Machine (IHM)**

Valentine a pris en charge le développement de **l’Interface Homme-Machine (IHM)** en utilisant **Angular** avec la dernière version disponible ainsi que la librairie **Angular Material** pour garantir une interface moderne. Son travail a été centré à la fois sur **l’ergonomie, la navigation fluide** et surtout la **sécurisation des interactions avec l’utilisateur** et avec l’API.

**Travaux réalisés :**

**Technologie utilisée**

* **Angular dernière version** pour profiter des ssdernières fonctionnalités et mises à jour de sécurité.
* **Angular Material** pour des composants d’interface fiables et standardisés.
* Création de **schémas temporaires** pour la mise en place des **composants** et leurs **interactions**

**Méthodes mises en place pour sécuriser l'application :**

**1. Authentification sécurisée via Auth Guard**

* Valentine a **mis en place un Auth Guard Angular**, une technologie permettant de contrôler l'accès aux routes de l'application.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

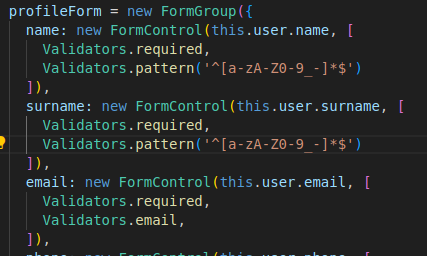
* Le système vérifie la présence d’un **access token valide** pour autoriser l’accès à la page principale du site ('/').
* **Si l’utilisateur n’est pas authentifié**, il est automatiquement redirigé vers la page de connexion.
* Ce contrôle permet également **d’éviter qu’un utilisateur déjà connecté ait à se reconnecter manuellement.**
* De plus nous ne pouvons pas avoir accès à des pages si nous n’y sommes **pas autorisés.**

**2. Gestion des tokens JWT**

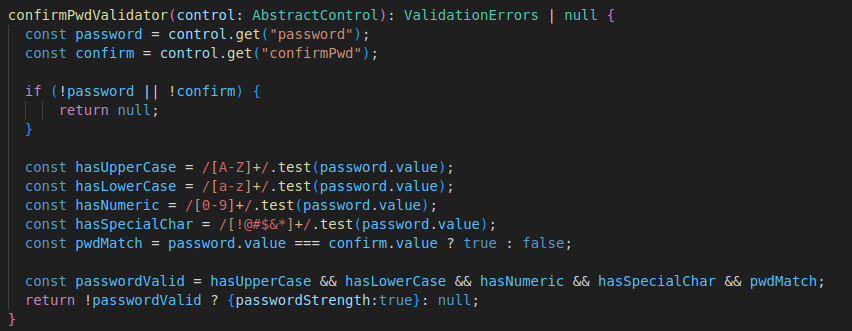
* Valentine a intégré la **gestion complète des access tokens et des refresh tokens** en lien avec l’API sécurisée.
* Elle a développé un **service Angular dédié** qui conserve et transmet correctement les tokens lors des appels API.
* Ce système garantit que **seuls les utilisateurs authentifiés peuvent interagir avec l’API** et que les sessions sont correctement sécurisées.
* L’**access token** est stocké dans le **local storage d’Angular**, que je gère via un service dédié (storage.service.ts). La gestion de l’api se fait également via un service dédié (api.service.ts).

**3. Protection des entrées utilisateur**

* L’IHM comporte plusieurs formulaires, notamment pour la connexion. Valentine a **sécurisé ces formulaires via les Validators Angular** qui contrôlent la conformité des données saisies avant de permettre l'envoi.



*Global*



*Personnalisé*

* Par exemple, il est impossible de soumettre des champs vides ou mal formaté.
* Après validation, **les données sont systématiquement "sanitisées"** grâce au DomSanitizer d’Angular pour empêcher l’injection de scripts ou l’envoi de données dangereuses vers l’API.
* Angular assure aussi une **protection native contre les attaques XSS** (Cross Site Scripting) notamment via la gestion automatique des URL et des attributs sensibles dans le DOM.

L’IHM comporte plusieurs formulaires, notamment pour la connexion. Valentine a **sécurisé ces formulaires via les Validators Angular**

**Choix techniques et sécuritaires :**

* **JWT + Auth Guard** : authentification robuste et contrôle des routes.
* **Formulaires protégés** : validation stricte et sanitation des entrées utilisateurs.
* **Services dédiés** : gestion centralisée et sécurisée des appels API et des tokens.
* **Chiffrement des canaux**: Les canaux vers l’api sont en https. On hash le mot de passe de l’utilisateur côté serveur pour réduire la surface d’attaque.

**Difficultés rencontrées :**

* Bien comprendre le système des tokens JWT (access + refresh) et gérer leur stockage de façon sécurisée.
* Adapter les Validators Angular pour couvrir tous les cas possibles et éviter les failles potentielles.

# vi- suard erwann

**Responsable : Sécurisation du canal de communication entre le client CLI et le serveur**

Erwann a développé un **serveur Python écoutant sur le port 8443**, utilisant un **canal TLS sécurisé** pour les échanges avec le client en ligne de commande (CLI). Son travail a principalement consisté à **mettre en place une authentification forte du serveur par le client** via un mécanisme de **challenge-réponse cryptographique**, tout en appliquant de nombreuses **bonnes pratiques de sécurité bas niveau.**

**Travaux réalisés :**

**Mise en place du serveur sécurisé**

* Création d’un **serveur Python écoutant sur le port 8443**, avec **ouverture d’un canal TLS sécurisé** dès la connexion.
* Lors de l’ouverture de la connexion, **le serveur transmet sa clé publique** au client.

**Mécanisme de challenge-réponse sécurisé**

* Le **client génère un aléa sécurisé** et l’envoie au serveur.
* Le **serveur signe cet aléa avec sa clé privée** et le renvoie au client.
* Le **client vérifie la signature** grâce à la clé publique reçue, s’assurant ainsi de l’identité du serveur.
* Une fois validé, **le client envoie un nouvel aléa signé par le serveur et le processus se répète** jusqu’à atteindre le nombre d’itérations souhaité (défini via un argument CLI).

**Méthodes de sécurisation mises en place :**

**1. Sécurisation des entrées utilisateur**

* Validation stricte des **arguments CLI** : contrôle de leur présence, de leur format et de leur parsabilité avant toute exécution.
* Aucun traitement n’est effectué sur des entrées non vérifiées.

**2. Génération d’aléa sécurisé**

* Utilisation de la fonction système **getrandom()**, directement connectée au **pool d’entropie du noyau Linux**.
* Cette méthode est plus sûre que la plupart des générateurs aléatoires classiques en C, car :
  + Elle **ne passe pas par le filesystem** (contrairement à /dev/urandom en lecture directe), ce qui réduit les vecteurs d’attaque.
  + Elle garantit des **aléas de haute qualité cryptographique.**

**3. Robustesse et contrôle des erreurs**

* **Vérification systématique des codes de retour** et gestion rigoureuse des erreurs mémoire et d’exécution.
* **Aucune suite d’exécution n’est possible en cas d’erreur critique.**

**4. Bonne gestion mémoire**

* **Nettoyage de la mémoire avant sortie du programme** pour éviter les fuites mémoire (memory leaks) et réduire l’exposition aux attaques post-exécution.

**Choix techniques et sécuritaires :**

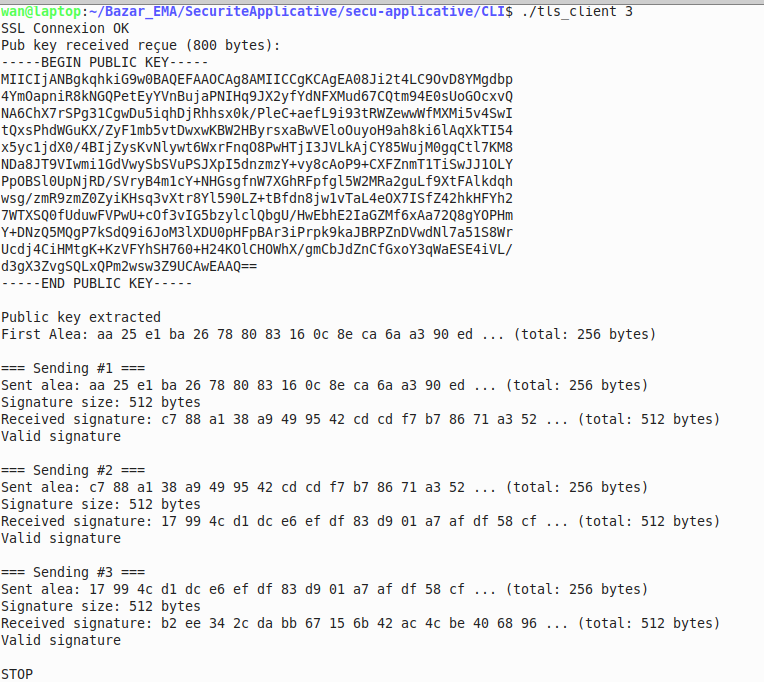
* **Vérification stricte des inputs dans le CLI** :  
  Validation des arguments et contrôle de leur parsabilité pour éviter tout comportement non prévu.
* **Utilisation de getrandom()** :  
  Fonction sécurisée qui puise directement dans le pool d’entropie du noyau Linux sans passer par le filesystem (réduit la surface d’attaque).  
  Source : [Man getrandom()](https://man7.org/linux/man-pages/man2/getrandom.2.html)
* **Vérification de tous les codes d’erreur** :  
  Chaque appel système ou fonction critique est vérifié. En cas d’erreur, l’exécution est immédiatement interrompue pour éviter tout comportement instable ou exploitation d’état inattendu.
* **Nettoyage de la mémoire avant de quitter le programme** :  
  Bonne gestion mémoire avec libération systématique des ressources utilisées, ce qui prévient les memory leaks et les failles potentielles.
* **Surveillance des interfaces ouvertes** :  
  Erwann a bindé le serveur sur INADDR\_ANY (écoute sur toutes les interfaces).  
  ➔ Cela **pourrait étendre la surface d’attaque**, mais la sécurité est renforcée par les **Network Policies Kubernetes** qui restreignent la communication aux services autorisés.

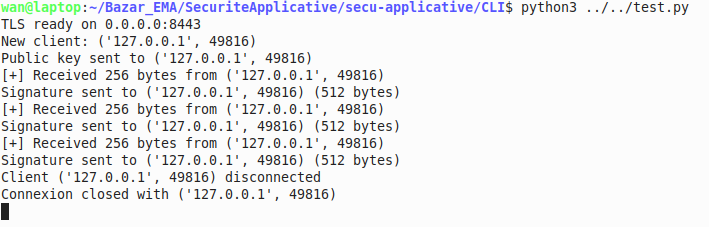
**Qualité de code et lisibilité**

* **Ajout de nombreux messages de debug** pour faciliter le suivi des échanges et le diagnostic des erreurs.
* **Commentaires précis et clairs** sur l’ensemble du code pour assurer la maintenabilité.

**Difficultés rencontrées :**

* Bien maîtriser la configuration des **canaux TLS en Python**.
* Implémenter correctement les vérifications de signature et gérer les erreurs de communication.
* Nettoyer correctement la mémoire dans un contexte sécurisé et répétitif (boucle d’aléas).

*Exemple côté client :*

*Exemple côté serveur :*

# vii- améliorations

**Julien**

* **Utilisation d’utilisateurs spécifiques aux droits limités** :  
  Bien que ce ne soit pas directement sa partie, il pourra collaborer avec l'équipe pour renforcer la sécurité des accès. Attribuer des comptes avec des permissions minimales pour chaque service ou composant permettrait de limiter l’impact potentiel en cas de compromission.
* **Renforcer les contrôles sur les quantités et la cohérence des données** :  
  Une vérification plus poussée des valeurs saisies et des seuils de cohérence permettrait de prévenir les abus et les incohérences dans le traitement des données.

**Romain**

* **Mise en place d’une limitation du nombre de requêtes (rate limiting)** :  
  Cela permettrait de protéger l’API contre des abus ou des tentatives de déni de service (DoS).
* **Amélioration de la lisibilité du code** :  
  Structurer davantage le code, ajouter des commentaires et des typages plus précis renforcerait la maintenabilité.
* **Enrichir les modèles Pydantic avec plus de contraintes et de validations** :  
  Préciser davantage les types, les plages de valeurs autorisées, et les longueurs maximales directement dans les modèles pour une validation plus stricte et plus sécurisée des données.

**Louis & Thomas**

* **Réflexion sur la haute disponibilité de la base de données avec des StatefulSets Kubernetes** :  
  Même si cela n’a pas pu être mis en place par manque de temps, nous avons envisagé l’utilisation de **StatefulSets** pour assurer la réplication et la persistance des données, ce qui aurait renforcé la tolérance aux pannes et la continuité de service.

**Valentine**

* **Ajouter une méthode de double authentification (2FA)** :  
  Cela aurait permis de renforcer significativement la sécurité de l’accès à l’application, notamment pour les comptes administrateurs.
* **Gestion des commandes et exploration d'autres types de sécurité (paiement, transactions, etc.)** :  
  La mise en place de flux sécurisés pour des fonctionnalités de gestion de commandes ou de paiements aurait permis de couvrir des cas d’usage plus sensibles et de tester d’autres mécanismes de sécurité avancés.

**Erwann**

* **Gestion des threads** :  
  Une amélioration envisagée serait de gérer les connexions simultanées via une gestion multithreadée optimisée, permettant d’améliorer la scalabilité et la robustesse du serveur en environnement de production.

# viii- conclusion

|  |  |  |
| --- | --- | --- |
| **Principe** | **Qui ?** | **Où et Comment ?** |
| **Intégrité** | Julien, Romain, Louis, Thomas, Erwann | - Base de données sécurisée et contrôlée (contraintes SQL, suppressions en cascade). - Validation des données via Pydantic. - Volumes persistants Kubernetes. - Mécanisme challenge-réponse assurant l’intégrité des échanges. |
| **Confidentialité** | Julien, Romain, Louis, Thomas, Valentine, Erwann | - Mots de passe hashés et stockés séparément. - TLS sur tous les canaux (API, CLI). - JWT avec Auth Guard sur l’IHM. - HTTPS via Let’s Encrypt. - NetworkPolicies restreignant les accès. |
| **Authentification** | Valentine, Erwann | - Authentification JWT via l’IHM (access & refresh tokens). - Auth Guard Angular protégeant les routes. - Authentification serveur par challenge-réponse CLI. |
| **Non-répudiation** | Erwann, Valentine | - Signatures numériques dans le protocole CLI. - Gestion des tokens JWT (traces des sessions). |
| **Fiabilité** | Louis, Thomas, Romain | - Redondance des pods Kubernetes. - Volumes persistants. - Validation stricte des entrées API (Pydantic). |
| **Disponibilité** | Louis, Thomas | - Orchestration Kubernetes avec redémarrage automatique. - Gateway sécurisée. - Multi-pods pour assurer la charge. |
| **Maintenabilité** | Romain, Valentine | - Code structuré (services dédiés, design patterns). - Documentation API claire. - Séparation des responsabilités. |
| **Sécurité** | Tous | - Application systématique des bonnes pratiques. - Stockage sécurisé des secrets. - NetworkPolicies. - TLS, HTTPS. - Validation et sanitation des entrées utilisateur. |

L'ensemble du projet a permis de mettre en place une infrastructure **sécurisée, fiable et maintenable**, répartie entre les différents membres de l’équipe, chacun ayant apporté des solutions techniques robustes :

* **Julien** a consolidé la sécurité des données au niveau de la base et a rédigé le rapport.
* **Romain** a sécurisé le backend et structuré le code pour assurer la fiabilité et la maintenabilité.
* **Louis et Thomas** ont construit une infrastructure Kubernetes hautement disponible avec une communication interne sécurisée.
* **Valentine** a protégé l’interface utilisateur en mettant en place une authentification forte et des protections anti-injection.
* **Erwann** a renforcé la sécurité des communications client-serveur avec un protocole cryptographique avancé.

Grâce à cette approche, le projet atteint **un niveau de sécurité global très élevé**, intégrant l’ensemble des bonnes pratiques modernes en matière de développement sécurisé et de gestion d’infrastructure cloud.