2025

MedicationLot – PharmaExpert

ROCKER BELL

Rédigé et réalisé par Rocker_Bell – Anass Tantane pour le Hedera Hackathon 2025

| rockbell8@gmail.com | testdevusecase@proton.me

https://github.com/rocker-bell/MedicationLot-Prototype

https://rocker-bell.github.io/MedicationLot-Prototype/

Sommaire:

Introduction

1.1 Présentation du projet – *PharmaExpert*

1.2 Objectifs et motivations

1.3 Utilisation de la technologie Hedera Blockchain

Architecture du système

2.1 Vue d'ensemble des phases fonctionnelles

2.2 Phase 1 – Authentification et gestion des crédits utilisateurs

2.3 Phase 2 – Traçabilité des lots de médicaments

2.4 Interaction entre les contrats intelligents

Description des contrats intelligents

3.1 Contrat `LotTracker` (MedicationLot)

3.2 Contrat `UserAuth` (IBA – Identity & Ban Administration)

3.3 Contrat `CreditToken` (ERC20)

Déploiement sur le réseau Hedera

4.1 Détails du réseau

4.2 Adresses et propriétaire des contrats

4.3 Intégration du token et compatibilité EVM

Logique fonctionnelle

5.1 Système de qualification et de frais (LotTracker)

5.2 Système de crédits utilisateurs (UserAuth ↔ CreditToken)

5.3 Sécurité et contrôle d'accès

Gestion des événements et traçabilité des données

6.1 Journaux d'événements et transparence blockchain

6.2 Transfert de lots et historique des propriétaires

6.3 Auditabilité et intégrité

Résumé et conclusion

7.1 Avantages du système

7.2 Limites et perspectives d'amélioration

7.3 Remarques finales

Description générale

Objectif du projet

Le projet **PharmaExpert** a pour objectif d'assurer une **traçabilité complète, fiable et infalsifiable** des médicaments, depuis leur **fabrication** jusqu'à leur **délivrance au patient final**.

Son but principal est de **lutter contre la contrefaçon** et les **anomalies logistiques** en garantissant la **transparence**, la **sécurité** et l'**intégrité des données** tout au long de la **chaîne d'approvisionnement pharmaceutique**.

Architecture fonctionnelle sur Hedera

Le projet s'appuie sur la **technologie blockchain Hedera Hashgraph**, qui offre **rapidité**, **frais de transaction minimes** et **haut niveau de sécurité**.
L'application se divise en **deux phases principales**, chacune gérée par un contrat intelligent distinct :

1. **Phase d'authentification et de gestion des utilisateurs**
   → Implémentée à l'aide des contrats **UserAuth** et **CreditToken**
   Ces contrats assurent :
   - L'inscription et l'identification des utilisateurs (pharmaciens, laboratoires, autorités, etc.)
   - La gestion des accès et des autorisations
   - L'attribution et le suivi de crédits numériques sur le réseau Hedera
2. **Phase de traçabilité des lots de médicaments**
   → Implémentée par le contrat **LotTracker (MedicationLot)**
   Ce contrat permet :
   - D'enregistrer chaque lot de médicament avec ses métadonnées (numéro de série, date, fabricant, etc.)
   - De suivre son parcours tout au long de la chaîne logistique (fabricant → distributeur → pharmacie → patient)
   - De garantir l'authenticité et l'intégrité des informations stockées sur la blockchain

En résumé

PharmaExpert combine les avantages de **Hedera Blockchain** avec une architecture **sécurisée et modulaire**, où :

- Les **contrats UserAuth et CreditToken** assurent la **fiabilité de l'identité numérique et de la gestion des droits**,
- Le **contrat LotTracker (MedicationLot)** garantit la **traçabilité immuable** des produits pharmaceutiques.

L'ensemble forme une solution complète de **confiance numérique** pour le secteur de la santé et de la distribution des médicaments.

Voici la description complète des fonctionnalités du contrat LotTracker - UserAuth - CreditToken:

LotTracker contract description :

Le contrat **LotTracker** est un suivi de chaîne d'approvisionnement basé sur des NFT (ERC-721).
Voici les principales caractéristiques et fonctionnalités :

- **Fonction principale** : Il crée (minte) un NFT unique (token ERC-721) pour chaque "lot de médicament" ajouté au système.
- **Propriétaire & Transfert** : Le propriétaire du NFT (ex : fabricant, distributeur, pharmacie) est le propriétaire actuel du lot. Le transfert du NFT via la fonction push représente un transfert de la garde dans la chaîne d'approvisionnement.
- **Stockage des données** : Toutes les données critiques (nom, fabricant, emplacement, statut) sont stockées directement sur la blockchain et liées à ce NFT unique.
- **Modèle économique** : Les utilisateurs (mais pas le propriétaire du contrat) doivent payer des frais en **CreditToken** (ERC-20) pour effectuer des actions.

🌐 Informations réseau

- **Blockchain** : Hedera Hashgraph
- **Service utilisé** : Hedera Smart Contract Service (HSCS)
- **Type contract** : ERC 721
- **Compatibilité** : EVM complète (supporte les contrats Solidity)
- **Déployeur / Owner** : 0x08a5b2500742b452f8e4ce60a3afaa96dcdd7a1b
- **Token associé** : CreditToken (contrat ERC20 également déployé sur Hedera)
- **Adresse** : 0xdB04e79caEa24AF20b4ad1AaAb0Ed2e67DCb9449

Système de Frais et Qualification à Deux Niveaux

C'est la logique métier la plus complexe et importante du contrat, destinée à garantir que les utilisateurs aient une participation minimale avant de pouvoir interagir avec le contrat et à leur faire payer une petite commission par transaction.

1. **Le CreditToken (ERC-20)**
   Le contrat est lié à un token ERC-20 externe (le **CreditToken**). Tous les frais sont payés en **CreditToken**, et non en HBAR ou ETH. Cela signifie que toutes les fonctions d'écriture (comme insert) ne sont pas payables en HBAR/ETH.
2. **Le Modificateur hasCredit (Qualification Initiale)**
   Ce modificateur est le "gardien" de toutes les fonctions principales. Il applique une vérification de "participation" une seule fois :
   - **Première Transaction** : Lorsqu'un utilisateur (qui n'est pas le propriétaire) essaie d'effectuer sa première action (par exemple insert), le modificateur vérifie si l'utilisateur a au moins **100 CreditToken** (minCreditBalance).
   - **Qualification** : Si l'utilisateur remplit la condition des 100 tokens minimum, le contrat définit isQualifiedUser[msg.sender] = true.
   - **Transactions Subséquentes** : Pour toutes les transactions futures, le modificateur vérifie que l'utilisateur est qualifié et passe outre la vérification des 100 tokens.
3. **La Fonction _takeFee (Frais par Transaction)**
   Cette fonction est appelée par toutes les fonctions principales des utilisateurs (comme insert, push, updateLotInfo) après que la vérification **hasCredit** ait été validée :
   - **Vérification des frais** : Elle vérifie si l'utilisateur dispose des tokens nécessaires pour payer le **fixedFee** (1 CreditToken).
   - **Approbation Requise** : Elle utilise creditToken.transferFrom(). Cela signifie que l'utilisateur doit avoir d'abord envoyé une transaction d'approbation au contrat **CreditToken**, autorisant le contrat **LotTracker** à retirer au moins 1 token.
   - **Exemption pour le Propriétaire** : Le propriétaire du contrat est exempté de la fonction _takeFee, ce qui lui permet de réaliser des actions sans payer de frais ni avoir besoin d'une approbation.

Résumé Fonctionnel Complet

Fonctions Principales (Transactions d'Écriture)

Ce sont les principales actions qu'un utilisateur qualifié peut effectuer. Toutes ces fonctions nécessitent le modificateur **hasCredit** et, sauf pour le propriétaire, appellent la fonction _takeFee.

Fonction	Description
insert(...)	Crée un nouveau lot. Minter un nouveau NFT (ID de Token) pour l'appelant (msg.sender) et stocke toutes les données fournies (nom, fabricant, date, emplacement, URI) dans le mapping medicationLots.
push(...)	Transfère un lot. Transfère le NFT du propriétaire actuel (msg.sender) à un nouveau propriétaire et met à jour currentLocation et currentHandler. Cela met aussi à jour ownershipHistory.
updateLotInfo(...)	Met à jour les informations du lot. Permet au propriétaire actuel d'un NFT de modifier ses métadonnées (nom, fabricant, emplacement, etc.).
setInactive(...)	Définit le statut du lot sur "Inactif".
activateLot(...)	Réactive un lot, modifiant son statut de "Inactif" à "Actif".
discardLot(...)	Définit le statut du lot sur "Discaré" (état final permanent).

Fonctions de Lecture (Vue)

Ces fonctions peuvent être appelées gratuitement pour récupérer des données depuis la blockchain.

Fonction	Description
pull(tokenId)	Récupère toutes les données du lot de médicament correspondant au tokenId (retourne toute la struct MedicationLot et l'historique des propriétaires).
getLotStatus(tokenId)	Retourne le statut du lot sous forme de chaîne (ex. : "Actif", "Inactif").
isQualifiedUser(address)	Retourne **true** ou **false** selon si un utilisateur a rempli la condition des 100 tokens.
medicationLots(tokenId)	Récupère la struct **MedicationLot** du mapping medicationLots.
ownershipHistory(tokenId, index)	Récupère l'historique des propriétaires d'un lot en fonction du tokenId et de l'index.
tokenIdCounter()	Retourne le nombre total de lots créés.
creditToken()	Retourne l'adresse du contrat **CreditToken**.
FEE_RECIPIENT()	Retourne l'adresse du portefeuille qui reçoit tous les paiements des frais fixes.
fixedFee()	Retourne le montant des frais pour chaque transaction (en wei).
minCreditBalance()	Retourne le solde minimum nécessaire pour la qualification (en wei).

Fonctions Administratives (Uniquement pour le Propriétaire)

Ces fonctions peuvent uniquement être appelées par le propriétaire du contrat (l'adresse qui a déployé le contrat).

Fonction	Description
setFixedFee(newFee)	Permet au propriétaire de modifier les frais fixes (1 token).
setMinCreditBalance(newMin)	Permet au propriétaire de modifier le solde minimum nécessaire pour la qualification (100 tokens).
setFeeRecipient(newAddress)	Permet au propriétaire de modifier l'adresse du portefeuille recevant les paiements des frais.
setCreditTokenAddress(newAddress)	Permet au propriétaire de mettre à jour l'adresse du contrat **CreditToken**.

Cela résume les fonctionnalités principales et administratives du contrat **LotTracker** ainsi que la logique des frais et qualification des utilisateurs.

UserAuth contract description :

Description générale

Le contrat **UserAuth**, également désigné comme **IBA (Identity & Ban Administration)**, est un **système d'authentification décentralisé** permettant la **gestion des utilisateurs**, de leurs **comptes**, et de leurs **crédits tokenisés**.

Il intègre la logique d'un **token ERC-20 externe** (comme CreditToken) pour gérer les crédits attribués aux utilisateurs et offre des fonctionnalités d'inscription, de connexion, de changement de mot de passe, et de bannissement.

Ce contrat a été déployé sur la blockchain Hedera, utilisant le **réseau Hedera Smart Contract Service (HSCS)** compatible **EVM (Ethereum Virtual Machine)**.
Cela lui permet de bénéficier de la **rapidité**, de la **sécurité hashgraph**, et de **frais de gaz extrêmement faibles**.

Fonctionnalités principales

1. Constructeur

constructor(address tokenAddress)

- Initialise le contrat avec l'adresse d'un **token ERC20** (ex : CreditToken déployé sur Hedera).
- Définit le **déployeur du contrat (msg.sender)** comme **propriétaire (admin)**.

Gestion des utilisateurs

Fonction	Description
register(string email, string passwordHash, string username)	Enregistre un nouvel utilisateur et lui attribue **100 crédits initiaux** + transfert de **100 tokens ERC20**.
login(string email, string passwordHash)	Permet à un utilisateur de se connecter après vérification du hash du mot de passe.
changePassword(string oldHash, string newHash)	Met à jour le mot de passe (stocké sous forme de hash).
getUserInfo() / getUserInfoByAddress(address)	Retourne les informations utilisateur (email, pseudo, date de création).
getAllUserInfo()	Liste tous les utilisateurs enregistrés avec leurs informations de base.

Crédits et tokens

Fonction	Description
requestCredits(uint256 amount)	Déduit un certain nombre de crédits internes d'un utilisateur.
checkUserCredit(address)	Vérifie le solde de crédits internes (non liés directement au token ERC20).
creditToken()	Retourne l'adresse du contrat ERC20 utilisé comme jeton de crédit.
checkTargetAddressTokenBalance()	Vérifie le solde du token ERC20 pour une adresse spécifique sur le réseau Hedera.

Administration et bannissement

Fonction	Description	Accès
banUser(address user)	Bannit un utilisateur du système.	Admin uniquement
unbanUser(address user)	Réactive un utilisateur précédemment banni.	Admin uniquement

Fonction	Description	Accès
getBannedStatus(address user)	Indique si un utilisateur est banni.	Public

Historique & journalisation

Fonction	Description
getLoginHistory(address user)	Retourne la liste des horodatages de connexion d’un utilisateur.
getMyLoginHistory()	Retourne l’historique de connexions de l’utilisateur actuel.
getCreationTime()	Retourne la date de création du compte connecté.

Événements

Événement	Description
Registered(address, string, string, uint256)	Nouvel utilisateur enregistré.
LoggedIn(address, uint256)	Connexion réussie.
CreditsDeducted(address, uint256)	Crédits internes déduits.
PasswordChanged(address, uint256)	Mot de passe modifié.
AccountBanned(address, address)	Compte banni par un administrateur.
AccountUnbanned(address, address)	Compte débanni par un administrateur.

🌐 **Informations réseau**

- **Blockchain** : Hedera Hashgraph
- **Service utilisé** : Hedera Smart Contract Service (HSCS)
- **Compatibilité** : EVM complète (supporte les contrats Solidity)
- **Type de contrat** :
- **Déployeur / Owner** : 0x08a5b2500742b452f8e4ce60a3afaa96dcdd7a1b
- **Token associé** : CreditToken (contrat ERC20 également déployé sur Hedera)
- **Adresse exemple** : 0xdbc22b309b0c46e43c08d39c7f8acf119e091651

Résumé

Le contrat UserAuth (IBA) sur **Hedera** agit comme un **système d’identité Web3 complet**, combinant :

- **Authentification décentralisée**
- **Gestion des comptes et crédits utilisateurs**
- **Intégration native d'un token ERC20 (CreditToken)**
- **Outils d'administration (ban/unban, logs, sécurité)**

Il est optimisé pour un environnement Hedera, garantissant :

- Des **frais minimes**
- Une **exécution rapide et fiable**
- Une **interopérabilité totale** avec l'écosystème EVM d'Hedera

Credit token contract description :

Description générale

Le contrat **CreditToken** est un **jeton ERC-20** standard basé sur la bibliothèque **OpenZeppelin**.
Il définit un **token fongible** avec un **nom**, un **symbole**, et une **quantité initiale** créée lors du déploiement du contrat.

Ce contrat a été déployé sur la blockchain Hedera, ce qui lui permet de bénéficier de la **vitesse**, des **frais très faibles**, et de la **sécurité** du réseau Hedera Hashgraph.

Ce token peut être utilisé comme :

- Monnaie numérique interne (crédits, points, récompenses, etc.)
- Jeton d'utilité dans une application décentralisée sur Hedera
- Actif interopérable compatible ERC20 pour les smart contracts EVM sur Hedera

Fonctionnalités principales

1. Constructeur

constructor(string memory name, string memory symbol, uint256 initialSupply)

- Initialise le nom et le symbole du token.
- Crée la **quantité initiale** (initialSupply) de tokens.
- Les tokens sont attribués directement au **déployeur du contrat** (msg.sender), c'est-à-dire le compte ayant déployé le contrat sur **Hedera**.

🌐 **Réseau**

- **Blockchain** : Hedera Hashgraph
- **Compatibilité** : EVM (Ethereum Virtual Machine)
- **Type de contrat** : ERC20 standard
- **Déployeur** : 0x0932c157d51c1a8b1df86de7a4cef9033567f684
- **Adresse exemple** : 0x42a05014306386b823329f777eb09ec1f493d69c

Exemple :

CreditToken("CreditToken", "CRD", 1000000);

→ Crée 1 000 000 tokens nommés *CreditToken (CRD)* pour le propriétaire.

2. Fonctions ERC20 standard

Fonction	Description	Type
name()	Retourne le nom du token.	view
symbol()	Retourne le symbole du token.	view
decimals()	Retourne le nombre de décimales (par défaut 18).	view
totalSupply()	Montant total de tokens en circulation.	view
balanceOf(address account)	Solde du compte donné.	view
transfer(address to, uint256 value)	Transfère des tokens de l'expéditeur vers un autre compte.	transaction
approve(address spender, uint256 value)	Autorise un tiers (spender) à dépenser une certaine quantité de tokens.	transaction
allowance(address owner, address spender)	Retourne la quantité que spender peut encore dépenser au nom de owner.	view
transferFrom(address from, address to, uint256 value)	Transfère des tokens depuis from vers to (utilise l'allocation).	transaction

3. Événements

Événement	Déclenché quand...	Champs
Transfer(address indexed from, address indexed to, uint256 value)	Des tokens sont transférés d'un compte à un autre.	from, to, value
Approval(address indexed owner, address indexed spender, uint256 value)	Une autorisation de dépense est modifiée.	owner, spender, value

4. Erreurs personnalisées (OpenZeppelin v5)

Le contrat inclut des **erreurs personnalisées** pour une gestion plus efficace du gaz et une meilleure lecture des échecs de transaction :

Erreur	Cause
ERC20InsufficientAllowance(spender, allowance, needed)	Le montant autorisé est inférieur au montant demandé.
ERC20InsufficientBalance(sender, balance, needed)	Le solde de l'expéditeur est insuffisant.
ERC20InvalidApprover(approver)	Adresse invalide passée à approve().
ERC20InvalidReceiver(receiver)	Adresse invalide passée à transfer().
ERC20InvalidSender(sender)	Adresse invalide passée à transferFrom() ou transfer().
ERC20InvalidSpender(spender)	Adresse invalide passée à transferFrom() ou approve().

Résumé fonctionnel

Fonctionnalité	Description	Accès
Création de tokens initiaux	Générés au déploiement pour le créateur.	Public
Transfert de tokens	Envoie de tokens entre adresses.	Public
Autorisation de dépenses	Permet à un tiers de transférer des tokens à votre place.	Public
Lecture des soldes et du total	Permet de connaître la quantité de tokens détenue.	Public (lecture seule)

Conclusion

Le contrat CreditToken est un **ERC20 simple et standardisé**, sécurisé par OpenZeppelin.
Il ne contient **aucune logique de gouvernance, de burn, de mint supplémentaire**, ni de mécanisme de taxes ou de frais — c'est un **token de base** idéal pour servir de jeton utilitaire ou de base à d'autres contrats.

Fiche de liaison – CreditToken ↔ UserAuth

Déploiement sur : Hedera Smart Contract Service (HSCS)

(Réseau compatible EVM – frais faibles, rapidité élevée, sécurité Hashgraph)

Vue d'ensemble

Le système est composé de **deux contrats interconnectés** :

Contrat	Rôle principal	Type	Déployé sur
CreditToken	Gère la création et le transfert des jetons ERC20.	ERC20 standard (OpenZeppelin)	**Hedera HSCS**
UserAuth (IBA)	Gère l'inscription, la connexion, les crédits et la réputation des utilisateurs.	Smart contract applicatif	**Hedera HSCS**

Ces deux contrats interagissent via l'adresse du CreditToken, transmise au constructeur du UserAuth lors du déploiement.

Flux d'interaction entre les contrats

Étape	Action	Contrats impliqués	Description
1	**Déploiement de CreditToken**	CreditToken	Le jeton CRD est créé et miné pour le déployeur (admin).
2	**Déploiement de UserAuth avec adresse du token**	UserAuth	L'adresse du CreditToken est passée en paramètre au constructeur :
			solidity constructor(address tokenAddress)
3	**Inscription d'un utilisateur**	UserAuth → CreditToken	Lors de register(), 100 crédits internes + **100 tokens CRD** sont transférés au nouvel utilisateur.
4	**Connexion / gestion du compte**	UserAuth	Vérification du hash du mot de passe, historique de connexion.
5	**Déduction de crédits**	UserAuth	Réduit les crédits internes (sans toucher au solde token).
6	**Bannissement / Débannissement**	UserAuth (admin)	L'administrateur (déployeur) contrôle les accès utilisateurs.

Liaison technique

Élément	Localisation	Description
ERC20 public creditToken;	UserAuth.sol	Instance du contrat CreditToken utilisée pour les transferts.
creditToken.transfer(msg.sender, 100 * 10 ** 18);	register()	Appel direct au contrat ERC20 pour attribuer 100 CRD.
checkTargetAddressTokenBalance()	UserAuth.sol	Lecture du solde d’un compte externe sur le contrat CRD.

Gestion des rôles et sécurité

Rôle	Permissions	Description
Owner (Déployeur)	Peut bannir/débannir des comptes, consulter les infos globales.	Défini automatiquement lors du déploiement (msg.sender).
Utilisateur	Peut s’inscrire, se connecter, demander des crédits, changer son mot de passe.	Restreint par le modificateur notBanned.
Contrat CreditToken	Agit comme source de tokens CRD.	Ne gère aucune logique utilisateur.

🌐 **Détails réseau (Hedera)**

Élément :	Valeur (exemple)
Réseau :	Hedera Smart Contract Service
Compatibilité :	EVM (Solidity 0.8.x)

Contrat Token (CreditToken) : 0.0.7129562

Contrat Utilisateur (UserAuth) : 0.0.7137887

Deployer :

- 0x08a5b2500742b452f8e4ce60a3afaa96dcdd7a1b (UserAuth)

- 0x0932c157d51c1a8b1df86de7a4cef9033567f684 (CreditToken)

📊 **Résumé du lien logique**

Fonction du système	Responsable	Dépendance
Création du token	CreditToken	-
Stockage des comptes utilisateurs	UserAuth	-

Fonction du système	Responsable	Dépendance
Attribution de tokens aux nouveaux utilisateurs	UserAuth	CreditToken.transfer()
Vérification du solde token	UserAuth	CreditToken.balanceOf()
Administration et sécurité	UserAuth	interne (owner)

🧩 En bref

Le système **IBA + CreditToken sur Hedera** forme une architecture complète :

- **CreditToken** = composant monétaire (ERC20 CRD)
- **UserAuth** = composant applicatif et identitaire
- Ensemble, ils permettent :
    - Une **gestion décentralisée des identités**
    - Un **système d'incitation via tokens CRD**
    - Une **interopérabilité totale** avec Hedera et les DApps compatibles EVM

Crédits

Rédigé et réalisé par @rocker_bell – Anass Tantane pour le Hedera Hackathon 2025