DApp Web3 de trading de ressources tokenisées (NFT)
- ResourceSwap -

Hauteurs :
Yousfi Mohamed Yassine
Mohamed Laachir

- 5BLOC -

Table des matières

1. Introduction

1.1 Contexte du projet

Le projet **ResourceSwap** est une application décentralisée (DApp) basée sur la blockchain Ethereum.
Son objectif est de tokeniser des ressources numériques sous forme de NFT et de permettre leur échange entre utilisateurs via un système d'offres, tout en appliquant des contraintes métiers stricts**.**

Le projet s'inscrit dans un contexte pédagogique visant à démontrer :

- La compréhension des concepts Web3,
- La capacité à concevoir des smart contracts sécurisés,
- La mise en place de règles métiers non contournables,
- L'utilisation de tests unitaires et de stockage décentralisé.

1.2 Cas d'usage choisi

Le cas d'usage retenu est un système d'échange de ressources numériques, inspiré de mécanismes de jeux de collection ou de type *Monopoly*.
Chaque ressource:

- Est unique,
- Posséde un niveau (*tier*),
- Appartient à un utilisateur identifié par son adresse blockchain,
- Peut-être échangée contre une autre ressource via un smart contract.

Contrairement à un simple transfert, l'échange est bilatéral et contrôlé, ce qui garantit :

- La propriété effective des actifs,
- La transparence,
- L'impossibilité de fraude.

1.3 Acteurs du système

Utilisateur :
Peut mint des ressources, consulter ses NFT, créer des offres et accepter des échanges.
Smart Contract ResourceSwap :
Garantit l'application des règles métiers et la sécurité des échanges.
Réseau Blockchain (Ethereum local – Hardhat) :
Assure l'exécution et l'immuabilité des transactions.
IPFS :
Stocke les métadonnées et images des ressources de manière décentralisée.

1.4 Scénario d'utilisation typique

1. Un utilisateur connecte son (MetaMask).
2. Il crée (*mint*) une ressource NFT avec :

    - Un nom,
    - Un type,
    - Un niveau (tier),
    - Une valeur,
    - Un lien IPFS vers les métadonnées.
3. Il crée une offre d'échange :
   Je propose mon NFT « A » contre ton NFT « B ».
4. Un autre utilisateur accepte l'offre.
5. Le smart contract:
    - Vérifie les règles,
    - Echange les propriétaires des tokens,
    - Applique un verrou temporel (*lock*).

2. Justification de l'utilisation de la blockchain

La blockchain est pertinente dans ce projet pour plusieurs raisons :

- **Propriété traçable** : chaque ressource appartient à une adresse unique.
- **Échanges « trustless »** : aucun tiers de confiance n'est nécessaire.
- **Historique immuable** : les transferts sont enregistrés on-chain.
- **Règles métiers non contournables**:appliquées directement dans le « smart contract ».
- **Décentralisation des données** : métadonnées stockées sur IPFS.

3. Architecture générale du projet

1.1 Stack technique

Blockchain / Smart Contract

- Ethereum (réseau local Hardhat)
- Solidity
- OpenZeppelin ERC721URIStorage
- ReentrancyGuard

Développement & tests

- Hardhat v3
- Mocha / Chai
- TypeScript

Stockage décentralisé

- IPFS (URI ipfs://)

Frontend

- React
- TypeScript
- Vite
- ethers.js v6
- MetaMask

1.2 Structure du dépôt

BLOCKCHAIN:

frontend/
resourceswap/
docs/

4. Smart Contract – Conception et choix techniques

4.1 Standard NFT et sécurité

Le contrat principal **ResourceSwap.sol** est basé sur :

- **ERC721URIStorage** : permet de lier chaque token à un URI IPFS.
- **ReentrancyGuard** : protège contre les attaques de réentrance.

4.2 Métadonnées on-chain

Chaque ressource NFT possède les champs suivants :

- name: nom de la ressource
- rtype: type de ressource
- tier: niveau (1 à 4)
- value: valeur libre
- ipfsUri: lien vers IPFS
- previousOwners[]: historique simplifié
- createdAt: timestamp de création
- lastTransferAt: timestamp du dernier transfert

4.3 Contraintes métiers implémentées

Contrainte	**Description**
MAX_OWNED = 4	Un utilisateur ne peut pas posséder plus de 4 ressources
COOLDOWN = 5 min	Délai minimum entre deux actions
LOCK_DURATION = 10 min	Verrou après action critique

4.4 Système d'échange

Le mécanisme d'échange fonctionne comme un mini-marketplace :

- CreateOffer (offeredTokenId, requestedTokenId)
- AcceptOffer(offerId)
- CancelOffer(offerId)

Avant toute action, le contrat vérifie :

- L'existence des tokens,
- La propriété réelle,
- Les approvals ERC721,
- Les contraintes temporelles.

5. IPFS – Stockage décentralisé

Les métadonnées des NFT sont stockées sur IPFS sous forme de JSON :

- Le tokenURI pointe vers ipfs://...
- Le frontend convertit l'URI via une gateway HTTP

Cela garantit:

- La décentralisation,
- L'intégrité des données,
- La conformité aux exigences du sujet.

6. Tests unitaires avec Hardhat

6.1 Organisation des tests

Deux fichiers principaux:

- ResourceSwap.mint.time.test.ts
- ResourceSwap.offers.cancel.test.ts

6.2 Cas testés

Les tests couvrent:

- Mint valide
- Dépassement de MAX_OWNED
- Cooldown non respecté
- Lock actif
- Création d'offre invalide
- Acceptation invalide
- Échange réussi
- Annulation d'offre
- Mise à jour des timestamps
- Sécurité des permissions

```
PS C:\Users\Laachir\Desktop\ECOLE\MASTER2\5BLOC\EVAL_V3\blockchain\resourceswap> npx hardhat test

Compiled 1 Solidity file with solc 0.8.28 (evm target: cancun)
No Solidity tests to compile

Running Solidity tests


Running Mocha tests


  ResourceSwap - Mint, Time rules and Limits
    ✓ mintResource: mint succeeds with correct tokenURI and metadata (987ms)
    ✓ Cooldown: createOffer twice without delay reverts with cooldown error (104ms)
    ✓ Lock: createOffer immediately after mint reverts with lock error (70ms)
    ✓ After lock duration, mint becomes possible again (71ms)
    ✓ MAX_OWNED: allows 4 mints and reverts on the 5th (94ms)
    ✓ tokensOfOwner returns all owned token IDs (84ms)
    ✓ Metadata timestamps: createdAt is less than or equal to lastTransferAt on mint (62ms)

  ResourceSwap - Offers / Accept / Cancel
    ✓ 1) createOffer succeeds after lock and creates an active offer with correct fields (89ms)
    ✓ 2) createOffer fails if offered token does not exist (78ms)
    ✓ 3) createOffer fails if requested token does not exist (75ms)
    ✓ 4) createOffer fails if caller is not owner of offered token (72ms)
    ✓ 5) createOffer fails if offered token is not approved (77ms)
    ✓ 6) acceptOffer fails if offer is inactive (92ms)
    ✓ 7) acceptOffer fails if acceptor is not owner of requested token (81ms)
    ✓ 8) acceptOffer fails if requested token is not approved by acceptor (87ms)
    ✓ 9) acceptOffer succeeds when both sides approved and swaps owners; offer becomes inactive (99ms)
    ✓ 10) cancelOffer can only be called by the offerer (86ms)
    ✓ 11) cooldown: createOffer then immediate createOffer fails (81ms)
    ✓ 12) lock: after acceptOffer, the user is locked and immediate action fails (97ms)


  19 passing (2s)


19 passing (19 mocha)
```

```
Saved html report to C:\Users\Laachir\Desktop\ECOLE\MASTER2\5BLOC\EVAL_V3\BLOCKCHAIN\resourceswap\coverage\html
| Coverage Report           |        |             |                 |
| ------------------------- | ------ | ----------- | --------------- |
| File Path                 | Line % | Statement % | Uncovered Lines |
| contracts\ResourceSwap.sol | 98.75  | 96.05       | 156             |
| ------------------------- | ------ | ----------- | --------------- |
| Total                     | 98.75  | 96.05       |                 |
```

6.3 Résultats

19 tests passants

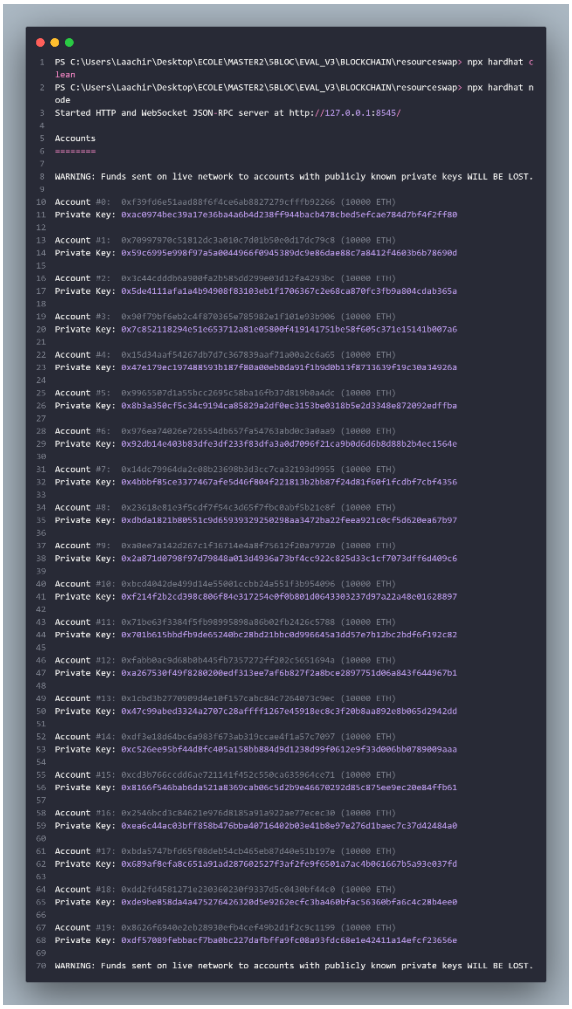
Coverage

- Lines ≈ 98.75 %
- Statements ≈ 96.05 %

7. Captures d’écran — Démonstration des règles métiers & gestion des erreurs

Remarque importante : les clés privées affichées par hardhat node sont uniquement destinées au réseau local de développement. Pour un dépôt public GitHub, il est recommandé de flouter/couper la partie “Private key” dans les screenshots.

Capture 01 — Hardhat Clean + Local Node



Cette étape permet de repartir sur une base propre et de lancer une blockchain locale : - npx hardhat clean supprime les anciens artefacts (cache / builds) pour garantir un environnement propre. - npx hardhat node démarre un serveur JSON-RPC local et affiche une liste de comptes de test (wallets) avec leur solde. Ces comptes servent à simuler plusieurs utilisateurs pour tester mint, approve, offers et swaps.

Capture 02 — Déploiement du contrat (deploy.ts)

Capture 03 — Mint d'un NFT ressource (mint.ts)

Cette étape montre la tokenisation d'une ressource : - Le script mint un NFT (ex: ANIMAL="Lapin"). - Il affiche le tokenURI stocké sur IPFS (ipfs://.../Lapin.json). - On voit le hash de transaction, le block, le tokenId créé et le propriétaire. Cela démontre : **tokenisation + métadonnées IPFS**.

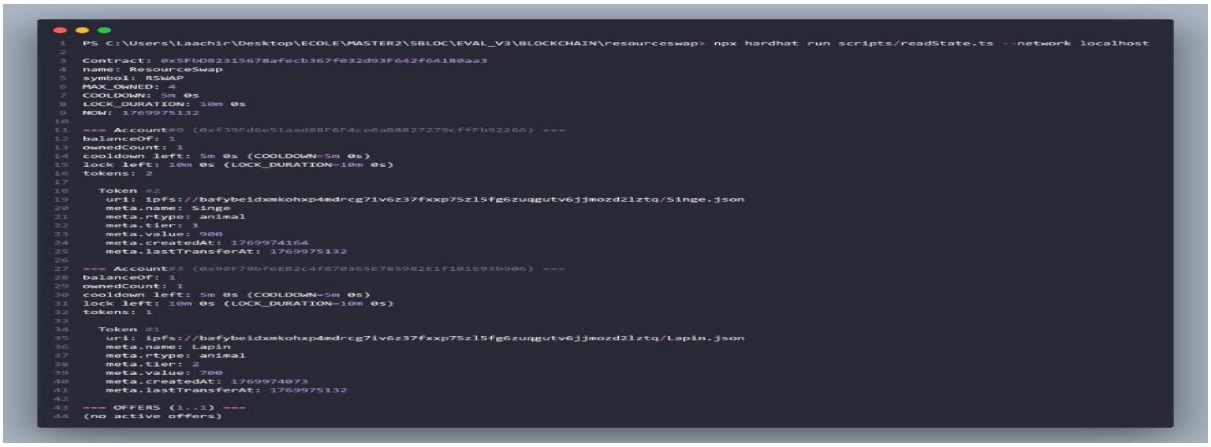
Capture 04 — Approve d'un token avant échange (approve.ts)

Avant qu'un token puisse être échangé, le propriétaire doit donner l'autorisation au contrat : - Le script affiche l'owner et l'id du token. - Une transaction approve est envoyée, permettant au contrat de transférer le token lors du swap. C'est un mécanisme clé : **aucun transfert possible sans consentement explicite**.

Capture 05 — CreateOffer bloqué par lock, puis résolu via warp (createOffer.ts + warp.ts)

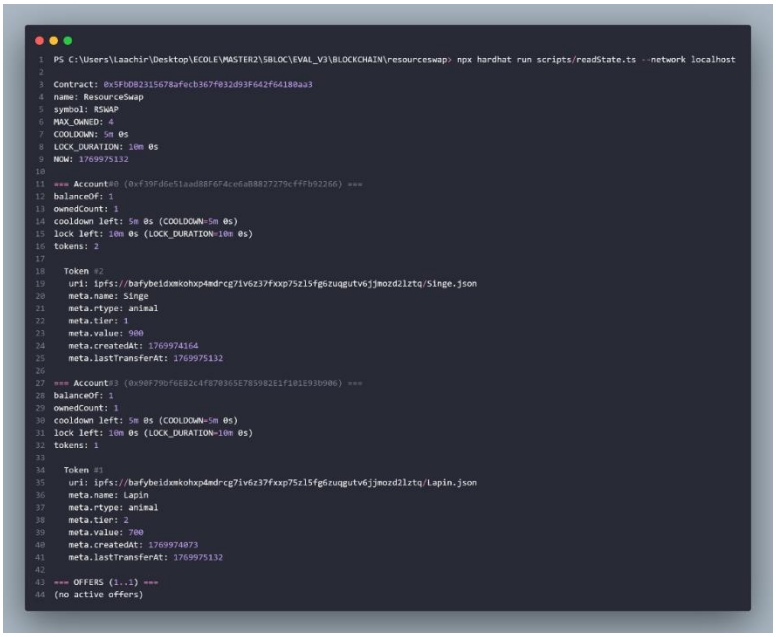
Cette capture illustre la contrainte temporelle **LOCK_DURATION** : - Une première tentative de createOffer échoue : l'utilisateur est encore verrouillé (lock active). - Pour tester ce comportement en local, on exécute warp.ts afin d'avancer le temps (ex: 620 secondes). - Après l'avance du temps, la même commande createOffer réussit et retourne un offerId. Cela prouve : **lock bien appliqué + outil de test du temps**.

Capture 06 — AcceptOffer échoue sans approve, puis réussit après approve



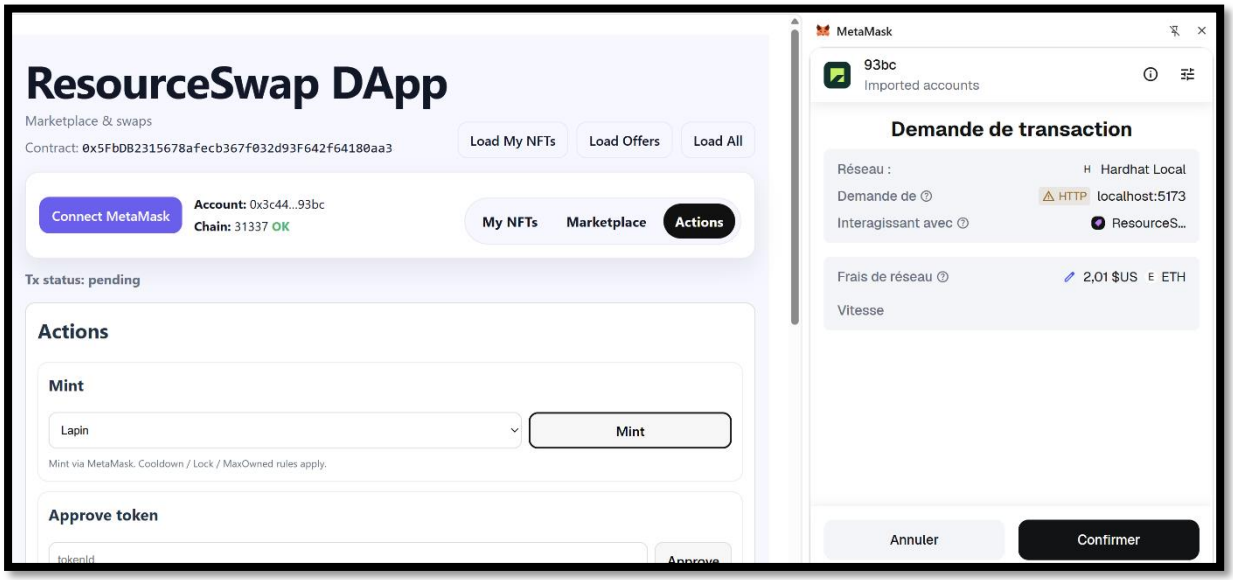
Cette capture montre une validation cruciale dans le swap : - La première acceptation échoue car le token demandé n'est pas approuvé pour le contrat. - Le script indique clairement la correction : exécuter approve.ts pour le token concerné. - Après approve, l'acceptation réussit et affiche les nouveaux propriétaires des deux tokens. Cela démontre : gestion d'erreur contrôlée + validation stricte avant swap.

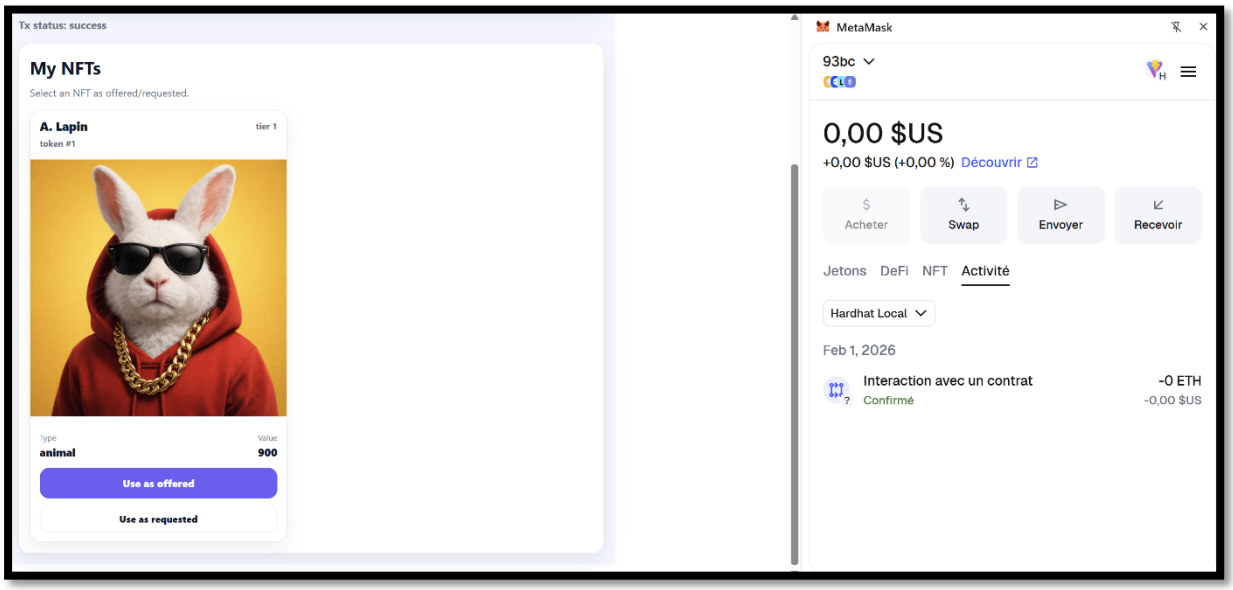
Capture 07 — Lecture de l’état complet (readState.ts)

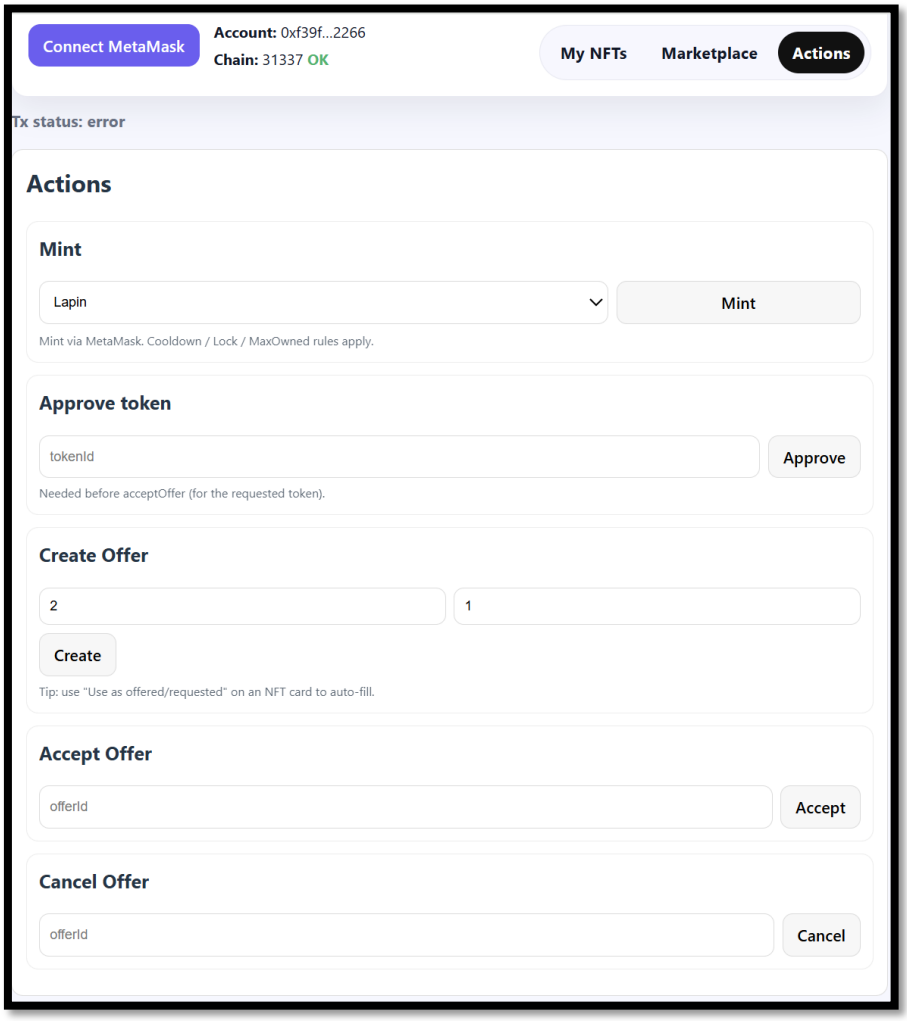


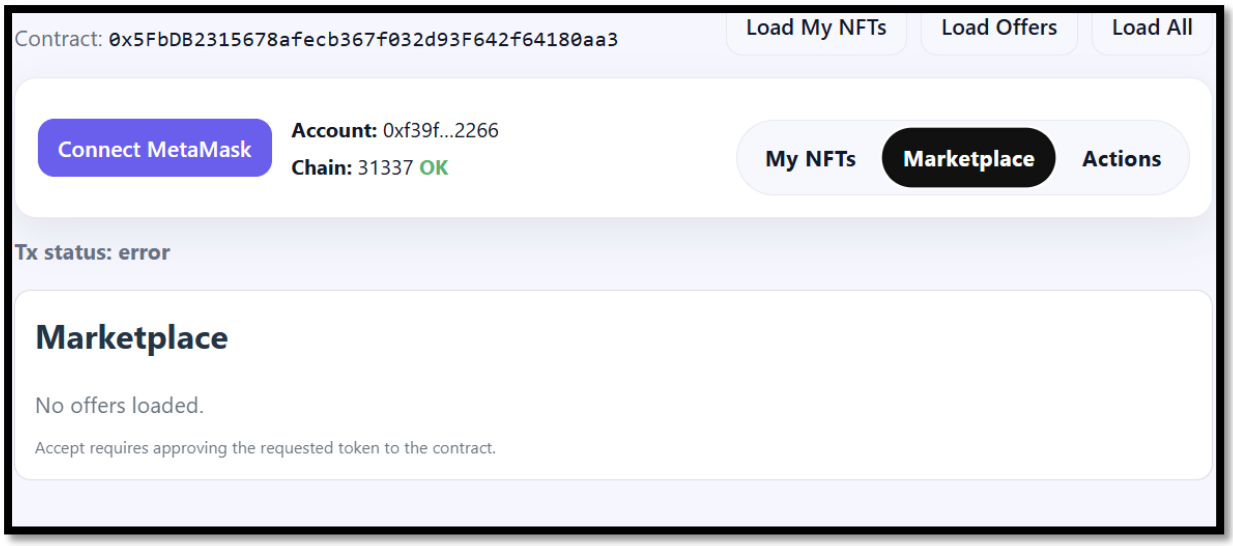
Le script readState.ts donne une vue claire de l’état du système : - Informations contrat : name, symbol, constantes (MAX_OWNED, COOLDOWN, LOCK_DURATION), timestamp actuel. - Par compte : nombre de tokens possédés, cooldown restant, lock restant, liste des tokens. - Par token : tokenURI IPFS, champs metadata (name/type/tier/value), timestamps. - Liste des offres actives (ou absence d’offres). Cette capture sert de preuve que **les données on-chain et les contraintes sont bien appliquées**.

8. Frontend DApp




Tx status: success
My NFTs
Select an NFT as offered/requested.
A. Lapin
tier 1
token #1
Type
animal
Value
900
Use as offered
Use as requested
MetaMask
93bc
0,00 $US
+0,00 $US (+0,00 %) Découvrir
Acheter
Swap
Envoyer
Recevoir
Jetons
DeFi
NFT
Activité
Hardhat Local
Feb 1, 2026
Interaction avec un contrat
Confirmé
-0 ETH
-0,00 $US


Connect MetaMask
Account: 0xf39f...2266
Chain: 31337 OK
My NFTs
Marketplace
Actions
Tx status: error
Actions
Mint
Lapin
Mint
Mint via MetaMask. Cooldown / Lock / MaxOwned rules apply.
Approve token
tokenId
Approve
Needed before acceptOffer (for the requested token).
Create Offer
2
1
Create
Tip: use "Use as offered/requested" on an NFT card to auto-fill.
Accept Offer
offerId
Accept
Cancel Offer
offerId
Cancel



9. Respect des exigences du sujet

Exigence	Réalisation
Tokenisation avec niveaux	NFT + tier
Échange de tokens	Offers + swap
Limite de possession	MAX_OWNED
Cooldown	COOLDOWN
Lock	LOCK_DURATION
IPFS	tokenURI
Tests unitaires	Hardhat
Couverture significative	>96%

10. Conclusion

Le projet **ResourceSwap** répond intégralement aux exigences du sujet et démontre une maîtrise solide des concepts fondamentaux du Web3.

Il met en évidence une conception rigoureuse de smart contracts sécurisés, une implémentation cohérente et non contournable des règles métiers (limite de

possession, cooldown et verrou temporel), ainsi qu'une architecture claire séparant la logique blockchain, l'interface utilisateur et la documentation. La qualité du projet est également renforcée par une validation approfondie via des tests unitaires couvrant l'ensemble des cas critiques, avec une couverture élevée attestant de la robustesse du contrat.

Enfin, bien que le projet soit fonctionnel et complet dans son périmètre actuel, des améliorations futures pourraient être envisagées, telles que la pagination des offres, l'indexation des données par « The Graph » ou encore un déploiement sur un testnet public afin de renforcer sa scalabilité et son réalisme.