

Rapport Technique - Card Collection DApp

1. Introduction

Ce projet consiste en une DApp de marché decentralisé de cartes de collection numériques. Les utilisateurs peuvent collectionner et échanger des cartes représentées sous forme de NFTs sur la blockchain Ethereum.

2. Choix techniques

Blockchain : Ethereum

Ethereum a été retenu pour son écosystème mature, le standard ERC-721 bien établi et la disponibilité d'outils comme Hardhat et OpenZeppelin.

Framework : Hardhat

Hardhat offre un environnement de développement complet avec compilation, tests et déploiement intégrés. Le plugin toolbox fournit les outils de test (Chai, Mocha) et d'interaction (Ethers.js).

Bibliothèques : OpenZeppelin

Les contrats OpenZeppelin sont audités et sécurisés. Nous utilisons :

- ERC721 et ERC721URIStorage pour les NFTs
- Ownable pour le contrôle d'accès
- ReentrancyGuard pour la protection contre les attaques de reentrancy

Stockage : IPFS via Pinata

Les métadonnées des cartes sont stockées sur IPFS pour garantir leur pérennité et leur décentralisation.

3. Architecture des contrats

CardNFT.sol

Contrat principal gérant les tokens ERC-721.

Structures de données :

- Enum **Rarity** : Commune, Rare, Epique, Legendaire
- Enum **CardType** : Guerrier, Mage, Creature, Artefact
- Struct **Card** : attributs complets de chaque carte

Fonctions principales :

- **mint()** : création d'une carte (owner only)
- **transferCard()** : transfert avec vérification des contraintes
- **getCard()** : lecture des attributs

CardMarketplace.sol

Contrat gerant les echanges entre utilisateurs.

Structure **Trade** : proposeur, receveur, tokens concernes, statut, timestamps.

Fonctions principales :

- **proposeTrade()** : proposition d'echange
- **acceptTrade()** : acceptation par le receveur
- **cancelTrade()** : annulation par l'une des parties

4. Respect des contraintes

Tokenisation des ressources

Les cartes sont des tokens ERC-721 avec 4 niveaux de rarete et valeurs associees :

- Commune : 10
- Rare : 50
- Epique : 200
- Legendaire : 1000

Echanges de tokens

Le systeme de proposition/acceptation permet les echanges directs. La regle de conversion impose un ratio maximum de 1:5 entre les valeurs des cartes.

```
uint256 public constant MAX_EXCHANGE_RATIO = 5;
```

Limites de possession

Chaque utilisateur est limite a 4 cartes :

```
uint256 public constant MAX_CARDS_PER_USER = 4;
```

Verification dans les fonctions **mint()** et **transferCard()**.

Contraintes temporelles

Cooldown de 5 minutes entre transactions :

```
uint256 public constant TRANSACTION_COOLDOWN = 5 minutes;
```

Lock de 10 minutes apres acquisition :

```
uint256 public constant ACQUISITION_LOCK = 10 minutes;
```

IPFS

Les metadonnees sont stockees sur IPFS. L'URI est passee lors du mint et stockee via `_setTokenURI()`.

Format des metadonnees :

```
{
  "name": "string",
  "type": "string",
  "rarity": "string",
  "value": "uint",
  "power": "uint",
  "defense": "uint",
  "description": "string",
  "image": "ipfs://...",
  "hash": "Qm...",
  "previousOwners": ["address"],
  "createdAt": "timestamp",
  "lastTransferAt": "timestamp",
  "edition": "uint",
  "maxEdition": "uint",
  "creator": "address"
}
```

Tests unitaires

47 tests couvrant :

- CardNFT : deploiement, mint, limites, cooldown, lock, transferts, lecture
- CardMarketplace : deploiement, propositions, ratio, acceptation, annulation, expiration, scenarios complexes

Execution : `npx hardhat test`

5. Securite

Patterns implementes

- **Checks-Effects-Interactions** : verifications avant modifications d'état
- **ReentrancyGuard** : protection contre les appels recursifs
- **Ownable** : restriction des fonctions administratives

Validations

- Verification de propriete des tokens avant transfert

- Controle des limites de possession
- Validation du ratio d'echange
- Verification des contraintes temporelles

6. Frontend

Application React avec :

- Connexion MetaMask
- Affichage de la collection personnelle
- Interface de proposition d'echange
- Panel d'administration pour le mint

Services :

- `wallet.js` : gestion de la connexion wallet
- `contracts.js` : interaction avec les smart contracts

7. Deploiement

Testnet Sepolia

Contrat	Adresse
CardNFT	0x8de801AE67Fb1E7Ed0Ac9E843a5c802413AE6d95
CardMarketplace	0xB72D7BA1524d6757B8b050fB57bfeb0Fcbe06A5C

8. Conclusion

Le projet repond a l'ensemble des contraintes du cahier des charges : tokenisation ERC-721, systeme d'echange avec regles de conversion, limites de possession, contraintes temporelles, stockage IPFS et tests unitaires. L'architecture separee en deux contrats permet une meilleure maintenabilite et une evolution independante de la logique NFT et marketplace.