# Développement et Déploiement d'un Contrat Intelligent de vente et d’échange d’images de debugging ducks

## Ce projet demande de mettre en œuvre beaucoup de connaissances. C’est pourquoi vous le réaliserez en binôme pour faciliter votre montée en compétences.

Vous disposerez d’un repository sur gitlab qui devra contenir toutes vos réalisations au fur et à mesure de vos avancements. J’attend au moins un commit par séance que vous taggerez avec un nom que je vous donnerai.

Dans ce repository, vous aurez un fichier **README.md** qui contiendra un compte rendu régulier et daté de ce que vous avez réalisé (un log).

Vous aurez également un fichier **INSTALL.md** qui expliquera la procédure d’installation de l’application et son fonctionnement.

### 1. Étape : Compréhension et spécifications

- **Objectifs** :
    - Comprendre les standards ERC-721 pour les NFT.
    - Identifier les fonctionnalités principales : création, mise en vente, achat, transfert des NFT.
    - Définir les rôles (vendeur, acheteur, propriétaire).
- **Sous-tâches** :
    - Étudier la norme ERC-721 : lire la documentation officielle sur EIP-721.
    - Définir les propriétés des NFT (e.g., URL des images des debugging ducks, propriétaire).
    - Concevoir une mini-spécification pour le contrat intelligent (quelles fonctions doivent exister ?).
- **Livrables** :

    - Un document ou une maquette listant les fonctionnalités du contrat et de l'interface utilisateur. (format markdown)
- **Critères d'évaluation** :
    - Clarté des fonctionnalités identifiées.
    - Pertinence des choix (par exemple, ajout de métadonnées comme le nom ou une description pour chaque NFT).

## 2. Étape : Développement du contrat intelligent en Solidity

- **Objectifs** :
    - Développer le contrat intelligent en Solidity pour gérer :
        - La création d'un NFT (mint).
        - La mise en vente d'un NFT.
        - L'achat/transfert d'un NFT.
    - Respecter le standard ERC-721.
- **Sous-tâches** :
    - Importer les bibliothèques OpenZeppelin (elles implémentent déjà ERC-721 de manière sécurisée).
    - Implémenter les fonctions principales :
        - `mint` : Création d'un NFT avec une image de duck (l'URL de l'image sera dans les métadonnées).
        - **Mise en vente** : Associer un prix à un NFT donné.
        - **Achat** : Permettre à quelqu'un d'acheter un NFT et de transférer la propriété.
    - Écrire des tests pour valider le contrat localement avec **Hardhat** ou **Remix**.
- **Livrables** :
    - Un contrat Solidity respectant ERC-721, commenté et testé.
    - Des scripts de déploiement avec Hardhat (ou un autre outil).
- **Critères d'évaluation** :
    - Fonctionnalités du contrat testées avec succès (tests unitaires sur Hardhat ou Remix).
    - Respect du standard ERC-721 (e.g., compatibilité avec OpenSea).

## 3. Étape : Création de l'interface Web3 avec Next.js ou React

- **Objectifs** :
    - Développer une interface utilisateur pour interagir avec le contrat.
    - Connecter la dApp au contrat intelligent via **Ethers.js** ou **Web3.js**.
- **Sous-tâches** :
    - Ajouter une fonctionnalité de connexion avec un portefeuille crypto (e.g., **Metamask**).
    - Afficher une liste des debugging ducks disponibles à l'achat (lire les données depuis la blockchain).
    - Permettre de :
        1. Mint un NFT (création).
        2. Mettre en vente un NFT.
        3. Acheter un NFT.
    - Ajouter un système de feedback utilisateur (e.g., messages de transaction réussie/échouée).
- **Livrables** :
    - Une application fonctionnelle avec Next.js/React.
    - Une intégration Web3.js ou Ethers.js pour interagir avec le contrat.
- **Critères d'évaluation** :
    - Interface utilisateur  réactive.
    - Bonne gestion des erreurs et des feedbacks.

## 4. Étape : Test et déploiement

- **Objectifs** :
    - Tester le contrat et l'application dans des environnements simulés (Rinkeby, Goerli, ou Sepolia).
    - Déployer le contrat sur un testnet Ethereum.
    - Héberger l'application avec un service comme Vercel ou Netlify (à voir)
- **Sous-tâches** :

    - Tester toutes les fonctionnalités sur un réseau de test Ethereum (e.g., mint, mise en vente, achat).
    - Déployer le contrat intelligent sur un réseau de test.
    - Héberger l'application frontend.
- **Livrables** :
    - Une application fonctionnelle sur un testnet.
    - Un lien vers l'application hébergée (frontend).
- **Critères d'évaluation** :
    - Fonctionnalités complètement testées et validées.
    - Application accessible depuis une URL publique.

## 5. Étape : Documentation et rapport final

- **Objectifs** :
    - Documenter le projet pour qu'il soit compréhensible par d'autres développeurs ou utilisateurs.
- **Sous-tâches** :
    - Rédiger une documentation pour le contrat (structure, fonctions, comment interagir avec).
    - Expliquer comment déployer et tester l'application.
    - Inclure une section sur les problèmes rencontrés et les solutions apportées.
- **Livrables** :
    - Un rapport final complet. (format MD)
    - Documentation technique et guide utilisateur. (format MD)
- **Critères d'évaluation** :
    - Documentation claire et complète.
    - Capacité à expliquer les choix techniques.

## 6. Mise à jour du plan avec l'intégration d'IPFS

- **Impact d'IPFS** :
    - Chaque NFT doit inclure un lien vers une image qui sera stockée sur IPFS.
    - Les métadonnées des NFT (en JSON) incluront l'URL IPFS de l'image.

- **Tâches supplémentaires** :
    - Étudier le fonctionnement d'IPFS et des services comme **Pinata** ou **Web3.Storage** pour héberger les images.
    - Identifier les métadonnées nécessaires pour chaque NFT (e.g., nom du duck, description, URL de l'image).
- **Critères d'évaluation** :
    - Les métadonnées des NFT sont correctement définies pour inclure l'URL IPFS.
- **Stockage des images**:
    - Stocker les images des debugging ducks sur IPFS et récupérer leurs **CID (Content Identifier)**.
    - Associer chaque NFT à son image via l'URL IPFS.
    - **Sous-tâches** :
        - Choisir un service IPFS :
            - **Pinata** : Plateforme avec une interface simple pour uploader et pinner les fichiers (nécessite un compte).
            - **Web3.Storage** : Une solution gratuite et rapide pour interagir avec IPFS.
            - **Infura** : Fournit une passerelle IPFS
    - Modification du contrat Solidity :
        - Modifier la fonction `mint` pour inclure un paramètre correspondant à l'URL des métadonnées (le CID IPFS).
    - Intégration avec le frontend :
        - Ajouter une option dans l'interface utilisateur pour uploader des fichiers sur IPFS (via Pinata/Web3.Storage API).
        - Assurer que les utilisateurs puissent voir les images des NFT via l'URL IPFS.
- **Livrables** :

    - Les images et métadonnées des debugging ducks sont disponibles sur IPFS.
    - Le contrat Solidity intègre les URL IPFS dans les métadonnées des NFT.

    - Le frontend affiche correctement les images depuis IPFS.
- **Critères d'évaluation** :
    - Les URL des images des NFT utilisent le protocole IPFS (`ipfs://<CID>`).
    - Les NFT affichent correctement leurs images et métadonnées dans l'interface utilisateur.

## Étape finale : Test des images et métadonnées

- Avant de déployer sur un testnet, assurez-vous que :
    1. Les images et métadonnées sont correctement uploadées sur IPFS.
    2. Les URL IPFS fonctionnent (tester avec `https://ipfs.io/ipfs/<CID>`).
    3. Les métadonnées des NFT affichent bien les images et attributs sur votre interface.

## Références utiles pour avancer :

1. **Solidity** :
    - Documentation officielle : https://soliditylang.org/.
    - Tutoriel pour les NFT avec Solidity et Hardhat : https://docs.openzeppelin.com/contracts/4.x/wizard.
2. **ERC-721 Standard** :
    - Explications détaillées : https://eips.ethereum.org/EIPS/eip-721.
    - OpenZeppelin ERC-721 implémentation : https://docs.openzeppelin.com/contracts/4.x/erc721.
3. **Intégration Frontend (Web3.js / Ethers.js)** :
    - Ethers.js : https://docs.ethers.org/v5/.
    - Metamask + React : https://docs.metamask.io/guide/create-dapp.html.

4. **Test et déploiement** :
    - Hardhat : https://hardhat.org/.
    - Déploiement sur un testnet Ethereum : https://ethereum.org/en/developers/tutorials/deploy-your-first-smart-contract/.
5. **Frontend frameworks** :
    - Next.js : https://nextjs.org/docs.
    - React : https://reactjs.org/docs/getting-started.html.

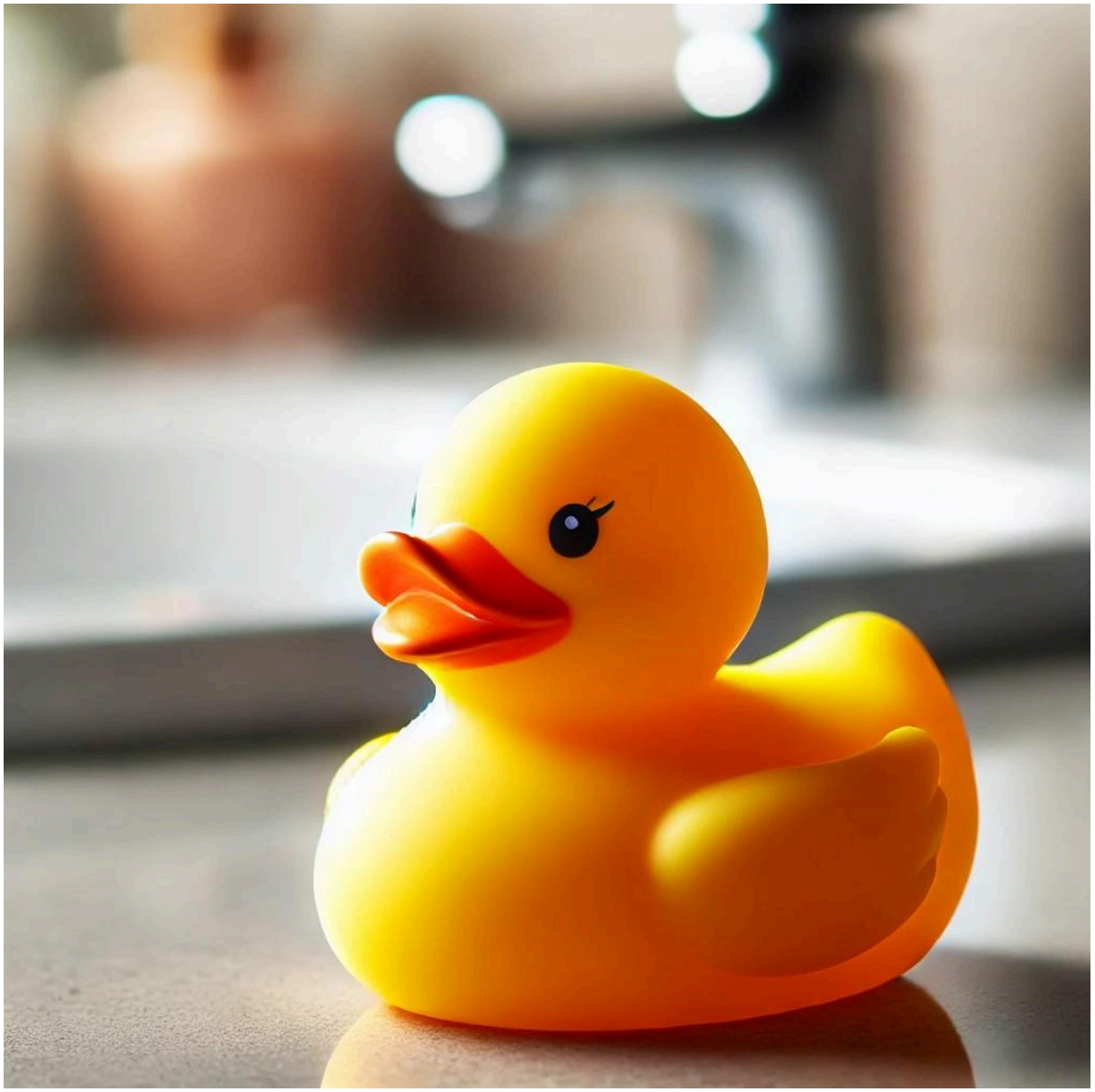
## Technologies et outils pour IPFS

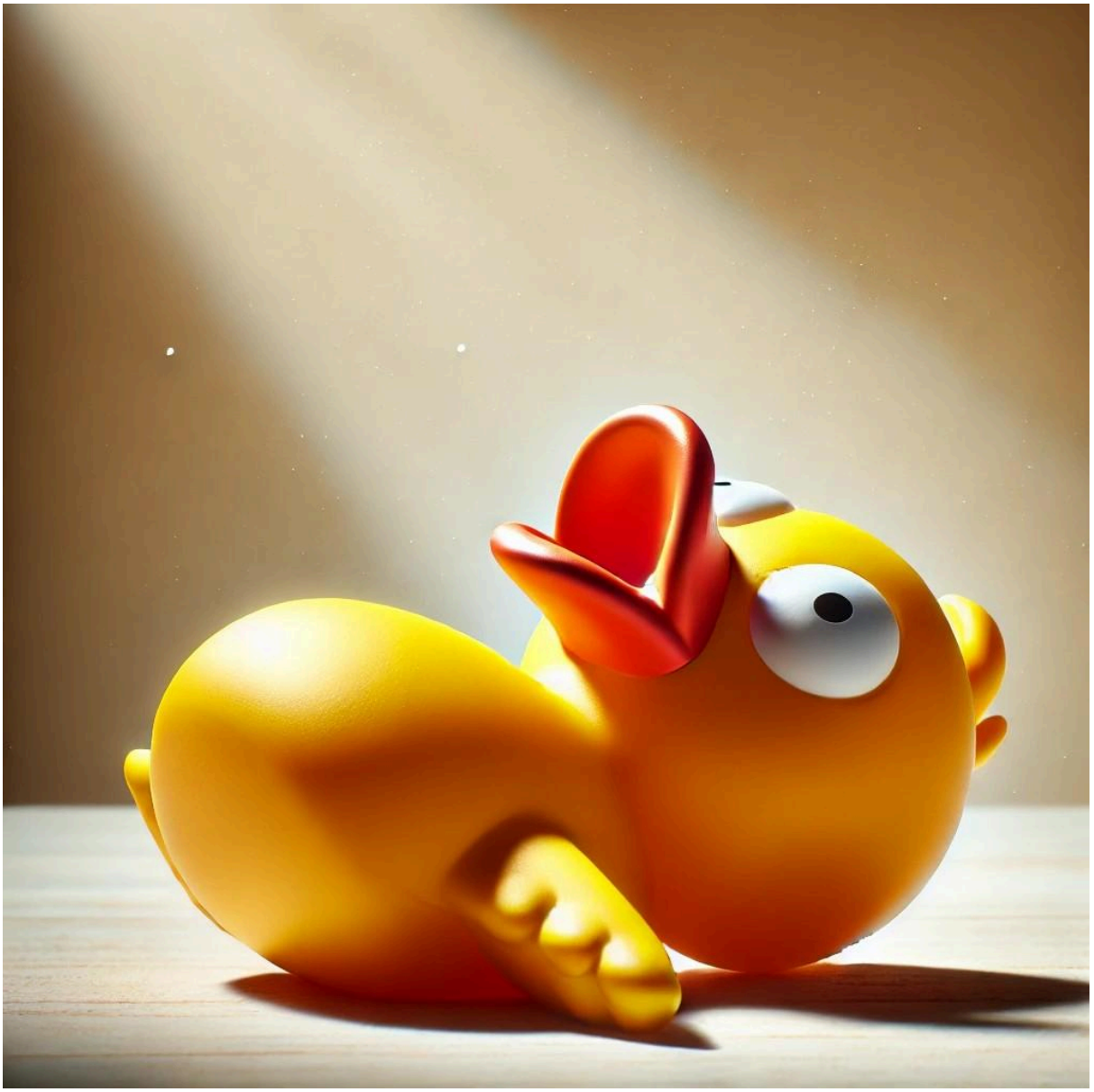
Voici quelques options pour travailler avec IPFS :

1. **Pinata** :
    - Site : https://www.pinata.cloud/
    - Documentation : https://docs.pinata.cloud/
2. **Web3.Storage** :
    - Site : https://web3.storage/
    - Documentation : https://web3.storage/docs/
3. **Infura IPFS** :
    - Site : https://www.infura.io/
    - Documentation : https://docs.infura.io/infura/networks/ipfs/how-to/
4. **IPFS CLI** (si vous voulez explorer IPFS localement) :
    - Installation : https://docs.ipfs.tech/install/
    - Commandes principales : `ipfs add <file>` pour uploader un fichier localement.

## Critères d'évaluation globaux :

- **Aspects techniques** :
    - Le contrat intelligent fonctionne correctement et respecte ERC-721.
    - Les transactions sur la blockchain sont sûres et sans bugs.
- **Interface utilisateur** :
    - Facilité d’utilisation, simplicité
- **Documentation** :
    - Explications claires pour la reprise ou l’extension du projet.






33


STUDENT


FIRE
FIRE