Rapport de projet : 2 – Jeu de cartes à collectionner.

Auteurs : BAZIL Olando

Table des matières

1 Présentation.

Le but de ce projet est de réaliser un jeu de cartes à collectionner de manière décentralisée sur Ethereum et donc de créer des NFT. Ce projet demande donc de comprendre les différentes parties du jeu à réaliser, à savoir la partie on-chain et off-chain, respectivement l'hébergement sur la blockchain et le frontend/backend du jeu. Cependant, nous n'allons pas créer le jeu dans son intégralité, en effet, nous nous concentrons uniquement sur la construction de la partie collection. C'est-à-dire, notre application doit pouvoir nous permettre d'acquérir des cartes numériques. Cela implique l'échange, le parcours des cartes sur un navigateur et l'achat de boosters. Les cartes seront représentées sous forme de NFT selon la norme ERC-721.

Pour ce faire, nous allons proposer de décomposer ce projet en 3 grands axes.

Tout d'abord, nous allons implémenter les contrats permettant de créer et gérer les collections et les NFT de notre jeu de cartes. Pour ce faire, nous allons utiliser Solidity afin de créer nos contrats sur la blockchain, et également ERC-721 qui sera utilisé en tant que NFT dans notre projet.

Ensuite, nous allons créer la partie frontend/backend afin de permettre d'interagir avec nos contrats, de sorte que nos utilisateurs puissent acquérir des NFT de nos cartes. La partie backend servira à stocker les données de nos NFT. Nous avons choisi d'utiliser React pour le frontend car il est simple d'utilisation à notre avis et que nous avons l'habitude de nous en servir. Pour le backend côté serveur, nous allons utiliser Express.js afin de pouvoir stocker les données de nos NFT. Cela nous semble être un choix judicieux étant donné que la base de données d'Express.js sera remise à zéro à chaque relancement de notre projet, tout comme la blockchain locale.

Enfin, nous avons choisi d'utiliser l'API du JCC Pokémon pour ce projet à intégrer.

Nous allons tout d'abord décrire en détail les différents contrats utilisés, puis les différentes pages et leurs fonctionnalités tout en expliquant leur lien avec le backend, et enfin je conclurai sur la réalisation du projet.

1.1 La partie on-chain.

Pour la partie on-chain nous allons utiliser des contrat Solidity afin de pouvoir effectuer les actions nécessaires dans notre système. Les contrats que nous utilisons sont relativement simples. Nous avons 3 contrats : Card, Collection et Main.

1.1.1 le contrat Card.

Dans le contrat Card, nous trouvons un contrat qui est responsable de la création et de la gestion des NFT représentant les cartes dans notre système. Il hérite, comme convenu, de la norme ERC-721, ce qui implique qu'il peut créer des jetons non fongibles. Chaque carte NFT est associée à un numéro permettant de l'identifier, un nom et une image. Ce contrat permet au propriétaire de transférer des cartes à d'autres utilisateurs, permettant ainsi l'échange entre amis de ses propres cartes, bien que nous ne nous en occupions pas.

1.1.2 le contrat Collection.

Dans le contrat Collection, nous trouvons une collection de cartes qui lui est associée. Cette collection est limitée en nombre, car nous désirons un nombre fini de cartes pour chaque collection. De ce fait, nous y ajoutons une variable correspondant au nombre de cartes courant et total, en plus du nom de la collection et de la liste de cartes. Ce contrat offre une fonction permettant de connaître le nombre de cartes qu'elle contient afin d'éviter les débordements, ainsi qu'une autre permettant d'ajouter une carte à sa collection si nbCarte $<$ CartesTotal.

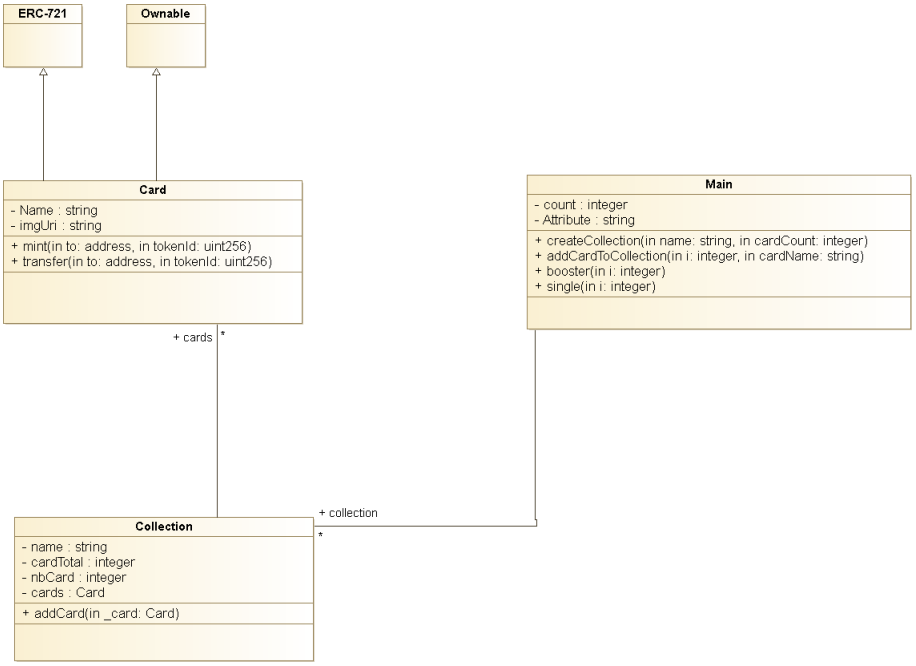

FIGURE 1 – Diagramme des contrats des compositions des contrats utilisés dans notre système et de leur interaction.

1.1.3 le contrat main.

Dans le contrat Main, ce contrat gère plusieurs collections de cartes et permet plusieurs actions, notamment la création de collections en spécifiant le nom de celle-ci et le nombre maximal de cartes. Il permet également l'ajout d'une carte à l'une des collections spécifiées, l'achat d'un booster de 10 cartes par un utilisateur, ainsi que l'achat d'un booster d'une carte par un utilisateur.

Ensemble, ces contrats interagissent pour créer un système permettant la gestion de collections de cartes numériques NFT sur la blockchain.

2 off-chain, le front et le back End.

Dans cette section, nous allons parler de la partie Off-chain, c'est-à-dire du front et du back.

2.1 Le front de notre application.

Le front de notre application est composé de 3 pages qui se relayent afin de permettre une expérience utilisateur adaptée aux besoins de ce projet. Afin de satisfaire ces derniers, nous avons opté pour 3 pages : une page d'accueil, une page de collection et une page de booster.

2.1.1 La page d'accueil.

La page d'accueil de notre site est relativement simple ; elle sert de point d'entrée à notre site. Elle sert également à récupérer le portefeuille de l'utilisateur actuel. Ainsi, lorsque l'utilisateur se connecte à notre application, le système enregistre son portefeuille afin de pouvoir utiliser les contrats de notre application.

2.1.2 la page collection.

La page de collection est naturellement plus chargée que la page d'accueil. Cette dernière permet de voir les NFT en possession de l'utilisateur actuel. Elle ne permet pas de voir la collection des 3-4 mois, car cela est géré dans le composant booster, comme nous le verrons dans sa section dédiée.

2.2 la page Booster.

La page Booster permet à l'utilisateur d'acheter un booster. Deux types de boosters sont disponibles à l'achat : un booster singulier contenant une carte ou un booster de 10 cartes. Nous avions initialement prévu de créer plusieurs collections, à la manière des gachas que l'on peut retrouver aujourd'hui. Par exemple, une bannière permanente contenant 140 Pokémon qui resteraient disponibles en permanence, et une autre bannière temporaire où l'on pourrait obtenir des Pokémon saisonniers. Cependant, en raison de contraintes de temps, nous avons finalement uniquement implémenté la bannière temporaire.

2.3 Le back de notre application.

3 Implémentation du serveur

Dans cette section, nous allons aborder l'implémentation du serveur de notre application. Nous couvrirons les données, l'API Rest utilisée et les liens entre le front.

3.1 Les données de la base de données

Avant de discuter des requêtes de notre API, commençons par présenter les données utilisées dans notre base de données. Nous en avons deux :

3.1.1 L'élément "Card"

"Card" est une structure comprenant un "TokenId," qui est un entier permettant d'identifier la carte, un "name," qui représente simplement le nom de la carte, et une "UriImg," qui est l'image de la carte. Étant donné que chaque carte "Mint" possède une adresse unique, cela signifie que nous n'avons pas besoin de créer 50 exemplaires de la même carte dans notre base de données. Par conséquent, nous avons choisi de conserver un seul exemplaire de chaque carte existante dans le système.

3.1.2 L'élément "Collection"

"Collection" est une structure comprenant un "tokenId" qui permet également de l'identifier, un "name" et une liste de cartes représentée par une liste de "tokenId," tous normalement uniques. Cela permet d'avoir une collection sans doublons, et donc lors de la création d'un booster, d'offrir la même chance pour toutes les cartes.

3.2 L'API

l'api de notre application nous permet plusieurs fonctionnalité réparti sur plusieurs route. Nous avons :

- **"/card"** Envoyer une requête post ici nous permet de créer une carte. Afin que la carte soit ajouté avec succès, il faut qu'elle ne soit pas déjà présente dans le système et que les 3 champs de la structure de données carte sois respecter dans le body.
- **/card/ :id** Permet de récupérer une carte selon son id.

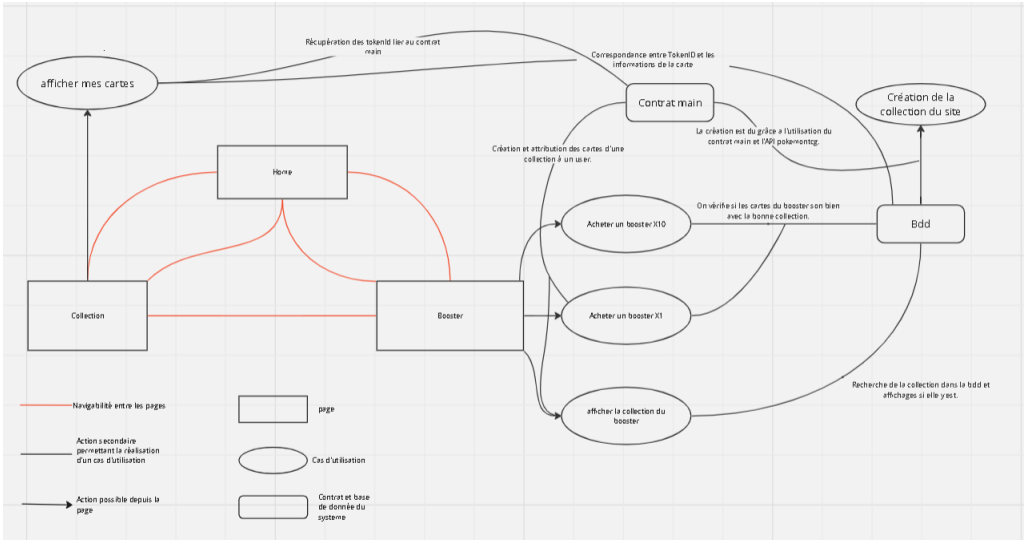

FIGURE 2 – Diagramme des contrats des compositions des contrats utilisés dans notre système et de leur interaction.

- /**collection** Envoyer une requête post ici nous permet de créer une collection. Afin qu'elle soit, ajouter avec succès le nombre de cartes total doit être > a 0, et le nom ne dois pas déjà figurer dans la bdd.
- /**collection**/ **:id** Permet de récupérer une collection selon son id.
- /**collection**/**card**/ **:id** Permet d'ajouter une carte à une collection selon l'id de la collection. Les conditions de succès sont les mêmes que pour /card. Cependant, il faut aussi que la carte ne soit pas déjà présente dans la collection.

3.3 Relation entre frond et back

Malheureusement, je n'ai pas réussi a lié le front et le back bien que je sais comment je devrais faire cependant en code, je n'y suis pas arrivé à temps.(avant la deadline) Voici donc comment j'avais imaginé le système.

Ainsi pour la page d'accueil on a cela.

pour la page collection, on aurait eu une récupération des tokenID du user par rapport au contrat main, puis fetch sur la bdd pour pouvoir afficher les images des NFT associer ce qui aurait lié la bdd au contrat main via le front.

Enfin pour les booster, on aurait eu une utilisation des contrat booster ou single ont fonction du bouton saisie par l'user. De là, il aurait payé avec sa cryptomonnaie la valeur associée au booster. De la 10 ou une carte aurait été ajouter au user qu'il aurait pu consulter dans la collection. Ou alors l'user aurait pu cliquer sur la collection du booster pour pouvoir voir toutes les cartes qu'il peut potentiellement avoir.

Dans le code un, user peut acheter un booster cependant, il ne peut pas la voir dans là sa collection à cause de la non-liaison des différents composants... Cependant, j'ai mis des cartes prédéfinies pour permettre la vision de l'affichage.

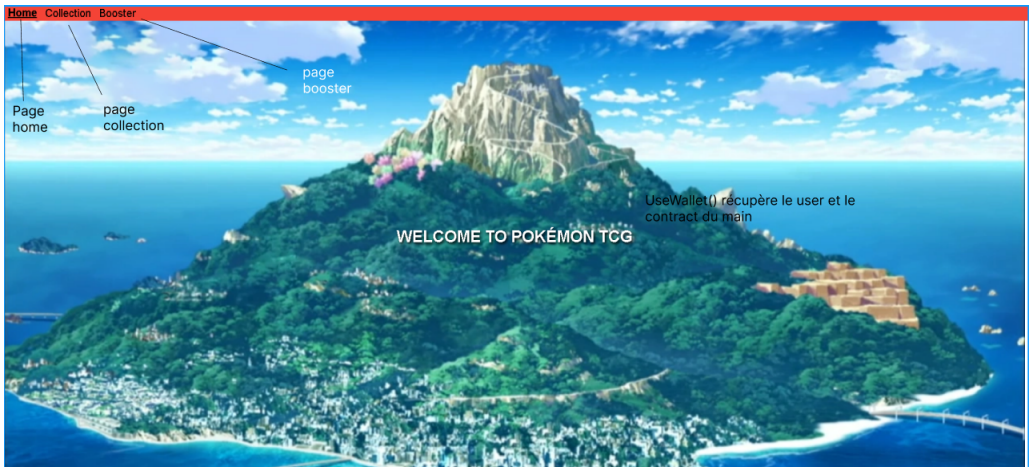

FIGURE 3 – Diagramme des contrats des compositions des contrats utilisés dans notre système et de leur interaction.

FIGURE 4 – Diagramme des contrats des compositions des contrats utilisés dans notre système et de leur interaction.

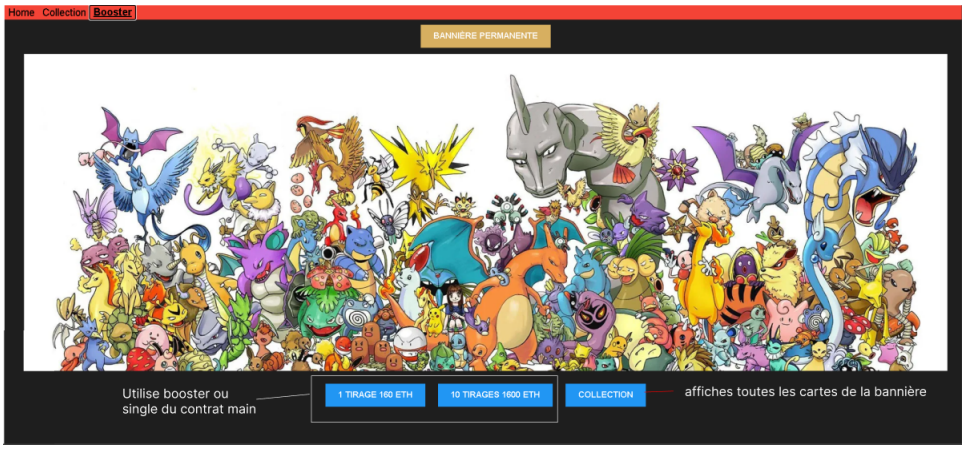

FIGURE 5 – Diagramme des contrats des compositions des contrats utilisés dans notre système et de leur interaction.

4 Difficulté rencontré

Plusieurs difficultés ont été rencontrées au cours de ce projet. Tout d'abord, nous avons été confrontés à de nombreux problèmes liés à l'installation des dépendances, et ces problèmes persistent encore aujourd'hui. Nous avons consacré beaucoup de temps à résoudre des erreurs de compilation dues à des versions de npm qui ne reconnaissaient pas OpenZeppelin, ainsi qu'à des erreurs de compilation des contrats, même ceux fournis sans modifications, qui signalaient que la limite de gaz était atteinte.

Nous avons également rencontré des problèmes de développement, tels que des bugs dans le développement du front-end, qui nous obligeaient à exécuter à nouveau "yarn dev" pour chaque modification, entraînant ainsi une perte de temps considérable, en combinaison avec des erreurs de connexion à MetaMask.

La compréhension du sujet s'est également avérée difficile en raison de l'ambiguïté concernant la limitation des NFT d'un côté et l'aspect "infini" du nombre de boosters que les utilisateurs peuvent acheter, du moins selon notre compréhension.

Au départ, nous pensions que nous devions créer une collection limitée de 10 000 cartes, avec une réduction à chaque booster. Cependant, à la fin de notre compréhension, il semblait que nous devions créer une collection représentant les 100 premiers Pokémon, que les utilisateurs pourraient obtenir de manière illimitée.

5 Conclusion

J'ai réussi à créer les composantes du projet séparément. Cependant, je n'ai pas réussi à les lier ensemble. Je peux toutefois fournir des points pour pouvoir y arriver.

Tout d'abord, il faudrait revoir la méthode booster de notre contrat. En effet, je me suis rendu compte que la nature du contrat Card impliquait de connaître l'URI de l'image avant la création de la carte. Cependant, lors d'un booster de la manière dont il est codé actuellement, on ne reçoit pas d'informations extérieures sur le contenu des cartes. Il faudrait donc avoir en paramètre deux listes, l'une de noms et l'autre d'images. Ainsi, on délègue l'aléatoire dans le front plutôt que dans le contrat, ce qui facilite les choses en plus de lier la base de données (BDD) et les NFT.

Il faudrait également récupérer les NFT de l'utilisateur actuel afin de pouvoir les afficher dans la collection via la BDD et le TokenID de tous les NFT possédés.