Master 2 : STL - Année 2020/2021

Projet n°2 DAAR : Decentralized-Wikipedia

Bello Pablo

Gomez Pierre

Enseignants :

Guillaume Hivert
Binh-Minh Bui-Xuan

6 décembre 2020

Table des matières

1 Introduction

Le but de ce projet de DAAR est d'implémenter une application web ressemblant à l'encyclopédie en ligne bien connue : *Wikipedia*. Il s'agira cependant de le faire d'une manière particulière, à l'aide d'une blockchain et de smart contracts, à l'image de la cryptomonaie *Ethereum*.
L'application web se fera à l'aide de *React* qui servira de front-end, les contrats seront écrit en Solidity et implémentés sur une blockchain de type Ethereum fournie par *Ganache*.
L'application demandée doit permettre d'écrire, de consulter et de modifier des articles qui seront stockés par transactions sur la blockchain.

2 Utilisation

Sur la page principale de l'application, on peut cliquer sur *Add an article* pour aller sur la page de création d'article ou sur *All articles* pour aller sur la page listant tous les articles.
Sur la page de création, il est possible de taper du texte dans la textarea. Le *MediumEditor* permet de selectionner le texte pour faire de la mise en forme. Il faut cliquer sur *Submit* pour envoyer l'article, *Metamask* s'activera pour demander une confirmation.
Sur la page listant les articles, on peut cliquer sur le lien *View* d'un article pour le consulter sur une page séparée.
Sur la page d'un article on peut cliquer sur le bouton *Edit* pour acceder à la page de modification de cet article.

3 Smart Contracts

Les Smart Contracts sont codés en Solidity qui est un langage permettant de définir des objets appelés contrats ainsi que les méthodes qu'ils promettent de réaliser.
Lorsque que le contrat est déployé sur la blockchain, il offre des structures de données modifiables par transaction.
Dans notre cas, le Smart Contract s'appelle Wikipedia et permet de gérer des articles. C'est donc un objet comportant deux structures données :
- *ids* : un tableau contenant les identifiants des articles (des entiers uint)
- *articlesById* : une map (table) qui associe à chaque identifiant un article
Un article est une structure contenant une chaîne de caractère.

Pour ajouter un article sur la blockchain, il faut créer un nouvel identifiant et l'ajouter au tableau, puis créer un article avec le contenu souhaité et l'insérer dans la map à la clé nouvellement créé :

```
function addNewArticle(string memory content) public {
  uint index = ids.length;
  ids.push(index);
  Article memory newArticle = Article(content);
  articlesById[index] = newArticle;
  }
```

Pour modifier un article, il suffit de créer un nouvelle article avec le contenu souhaité, et de l'insérer dans la map à la même clé qu'était l'ancien article, ce qui l'écrasera :

```
function updateArticle(uint index, string memory content) public {
    Article memory updatedArticle = Article(content);
    articlesById[index] = updatedArticle;
  }
```

Pour consulter un article, il suffit d'accéder à l'article correspondant à l'identifiant dans la map :

```
function articleContent(uint index) public view returns (string memory content) {
    return articlesById[index].content;
  }
```

On peut remarquer que, contrairement à addNewArticle et updateArticle, la fonction *articleContent* ne fait que lire le contenu de la map mais ne modifie pas ce qui est stocké sur la blockchain : elle porte donc la mention *view* dans sa déclaration. Ce faisant, cette méthode ne fera pas l'objet d'une transaction sur la blockchain. Cela se révelera important lorsqu'il s'agira d'appeler la methode du contrat en React.
Ces opérations seront appelées sur la blockchain, il ne faut pas oublier qu'il s'agit d'opérations asynchrones, en effet, nous sommes sur le réseau, de plus le minage d'un bloc confirmant leur réalisation peut prendre du temps. Elle feront donc l'objet de futures et de promises lors de leur appel.

4 React

4.1 NewArticle

Pour la création d'un nouvel article, la page comporte un formulaire composé d'une *textarea* à laquelle est affecté un éditeur de texte *MediumEditor*. Lors de la soumission du formulaire, un handler est appelé, il se charge de recupérer le contenu de la textarea via *editor.getContent()*. On verifie que la connexion au contrat a bien été établie avant d'appeler la méthode d'ajout d'article :

```
contract.methods.addNewArticle(DOMPurify.sanitize(editor.getContent())).send()
```

Pour des raison de sécurité et de confort, il est souhaitable d'empécher les utilisateurs d'ajouter des scripts, des medias, des redirection etc, dans le contenu des articles. Nous utilisons donc DOMPurify afin de nettoyer les données html contenues dans la textarea. Normalement, il convient d'appliquer cette sécurité lors de l'affichage du code HTML, ce que nous avons également fait par principe. Cependant, nous avons fait le choix de nettoyer le HTML avant même de le stocker dans la blockchain, car il nous paraissait plus judicieux de ne pas enregistrer des

données potentiellement malveillantes.

addNewArticle est une méthode qui modifie les données stockées sur la blockchain. Son appel donne lieu à une transaction, qui nécessite des frais en *gas* pour être réalisée. Il faut donc utiliser send() pour demander la transaction, qui transitera d'abord par le porte-monaie *MetaMask* pour être confirmée avant de l'executer sur la blockchain.

4.2 AllArticle

Pour lister tous les articles contenus dans la blockchain, on recupère tous les identifiants d'article avec l'appel sur le contrat :

```
1 contract.methods.getAllIds().call().then( ids => {...})
```

Cette méthode ne fait que lire des données, on utilise donc *.call()* pour l'appeler. Il faut noter qu'étant, asynchrone, elle envoie une promise, qu'il faut traiter dans un bloc *then*. On récupère donc un tableau d'indentifiant sur lequel on itère pour construire un tableau d'articles en appelant cette méthode pour chaque identifiant.

```
1 contract.methods.articleContent(i).call()
```

Pour l'affichage, on applique la fonction map sur le tableau d'articles pour créer une * * pour chaque article.

4.3 ReadArticleById

Pour afficher un article par identifiant, on verifie d'abord auprès du contrat que l'identifiant existe, puis on réclame le contenu de l'article associé. Si l'identifiant n'existe pas, on remplace l'affichage par une div *Not Found*.

4.4 UpdateArticle

Pour modifier un article, l'affichage ressemble à celui de la création d'un article seulement la *textarea* est pré-remplie avec l'article existant.
On vérifie que l'identifiant fourni existe bien dans la blockchain, puis on demande

l'article associé et on le stocke dans l'éditeur affecté à la *textarea*.
La mise à jour d'un article modifie les données donc donnera lieu à une nouvelle transaction sur la blockchain.

5 Difficlutés rencontrées

Plusieurs éléments nous ont posé problème lors de la réalisation de ce projet :
- Ganache n'affiche pas le contenu de la map des articles. Ceci nous a laisser penser que le contrat ne fonctionnait pas correctement et que la map des articles ne contenait rien. Or il est quand même possible d'utiliser la map.
- Nous avons pris un peu de temps à réaliser que le type de retour des fonctions des contrats sont des promises asynchrones et non pas directement la valeur attendue.
- Des difficultés à changer de page de l'application en React et à faire des chemins dynamiques en fonction des id des articles et de passer des paramètres.
- Après quelques recherches, il nous a semblé, sauf erreur de compréhension, que l'utilisation de Magic (Fortmatic) nécéssitait l'utilisation de vrai Ethereum pour fonctionner.

6 Conclusion

Ce projet n'est qu'un ébauche d'une vériatble application, la partie design ayant par ailleur été laissée de côté. Il nous a tout de même permis d'utiliser plusieurs nouvelles technologies, notamment les Smart Contracts en Solidity, déployés sur Ganache. Il a également été l'occasion d'étendre nos connaissance sur React.