Decentralized Wikipedia

Auteure : Mylène Balech
Date : Lundi 23 novembre 2020

Présentation
Le but de ce projet est d'implémenter un wikipédia décentralisé, en utilisant la blockchain pour stocker nos articles.
Cette application est développée en react JS pour la partie front-end et Ethereum est utilisé pour créer des contrats.

Table des matières

I. Le contrat

a. Création d'un article

```
//Création d'un nouvel article
function createArticle(string memory contenu) public{
  Article memory newArticle = Article(contenu);
  uint  index = ids.length;
  ids.push(index);
  articlesById[index] = newArticle;
  idsCountHistory[index] = 0;
}
```

Figure : Création d'un article

Nous créons un nouvel article en appelant la structure Article. Puis nous créons un nouvel index pour identifier de manière unique l'article. Cet index est stocké dans un tableau.

Enfin, nous créons le nouvel article dans le tableau contenant tous les articles, et définissons aussi la taille de son historique qui est de 0 au départ.

b. Modification d'un article

```
//Modification de l'article en fonction de son identifiant
function modifyArticle(uint id,string memory contenu) public{
  //Enregistrement de l'historique
  uint index = idsCountHistory[id];
  articlesHistory[id][index] = articlesById[id];
  idsCountHistory[id] = index +1;

  Article memory newArticle = Article(contenu);
  articlesById[id] = newArticle;
}
```

Figure : Modification d'un article

Afin de modifier un article, l'identifiant de celui-ci et le nouveau contenu sont passés en paramètre.

Le tableau contenant tous les historiques d'un article est de type mapping(uint => mapping(uint => Article)) , il permet de contenir pour un article son ensemble d'historique.

Nous enregistrons dans un premier temps son ancien état dans le tableau articlesHistory, à l'aide de son identifiant et du nombre d'historique que l'article a déjà eu.

Puis nous modifions ensuite l'article en créant un nouveau contenu.

c. Récupération des articles

En Solidity, c'est impossible de retourner un mapping d'élément, il faut passer par un identifiant et retourner le contenu du mapping en fonction de l'identifiant transmis.

Pour récupérer les articles, il faut donc dans un premier temps récupérer tous les identifiants.

```
//Récupère tous les ids
function getAllIds() public view returns (uint[] memory) {
  return ids;
}
```

Figure : Récupération de tous les ids

Puis lorsque tous les identifiants sont récupérés, il faut récupérer le contenu d'un article en fonction de son identifiant.

```
//Permet de récupérer le contenu d'un article en fonction de son identifiant
function articleContent(uint index) public view returns (string memory) {
  return articlesById[index].content;
}
```

Figure : Récupération du contenu d'un article

d. Récupération de l'historique d'un article.

Le fait de récupérer l'historique est le même procédé que pour récupérer le contenu d'un article.

Tout d'abord il faut récupérer le nombre d'historique enregistrées pour l'article.

```
//Retourne le nombre d'historique créé pour l'article
function getNumberHistorical(uint id) public view returns (uint){
  return idsCountHistory[id];
}
```

Figure : Récupération du nombre d'historiques en fonction de l'identifiant

Puis lorsque le nombre d'historique est récupéré, il suffit de récupérer l'ancien contenu grâce à son identifiant et de son numéro d'historique.

```
//Retourne l'historique d'un article en fonction de l'identifiant de l'article et du numéro de l'historique
function getHistorical(uint id, uint num) public view returns (string memory){
  return articlesHistory[id][num].content;
}
```

Figure : Récupération de l'historique de l'article.

II. Récupération des données via Ethereum

La méthode send() permet d'écrire dans le contrat, alors que la fonction call() permet de récupérer des éléments du contrat.

a. Ajout d'un article

Afin d'ajouter un article, nous devons envoyer le contenu au contrat. Ceci se fait en appelant directement la fonction du contrat permettant d'insérer l'article.
Cela se fait de cette manière : contract.methods.createArticle(contenu).send() ;

```
const saveArticle = (article) => async dispatch => {
  if(article != null) {
    const { contract } = store.getState() // récupération du contract
    await contract.methods.createArticle(article.toString()).send(); // création de l'article dans le contrat

    // Récupération de tous les articles pour mettre à jour suite à l'insertion du dernier
    const allarticles = getAllArticles(contract);
    dispatch(getAllArticlesStore({ allarticles })); // modification de l'état
  }
}
```

Figure : Création de l'article

b. Modification d'un article

Pour modifier un article, il suffit d'appeler la méthode correspondante du contrat en envoyant en paramètre l'identifiant de l'article et le nouveau contenu de celui.
Cela se fait de cette manière : contract.methods.modifyArticle(id,contenu).send() ;

```
const modifyArticle = (id,article) => async dispatch => {
  const { contract } = store.getState() // Récupération du contrat

  // Envoie de la mise à jour
  await contract.methods.modifyArticle(id,article.toString()).send(); // Envoie de la modification du contenu

  // Mise à jour des articles dans l'application
  const allarticles = getAllArticles(contract);
  dispatch(getAllArticlesStore({allarticles})); // Mise à jour de l'état

  // Mise à jour de l'historique de l'application
  const historical = await getHistorical(contract);
  dispatch(updateHistorical({historical})); // mise à jour de l'état
}
```

Figure : Modification d'un article

c. Récupération des articles

Pour récupérer les articles, il faut appeler la fonction correspondante.
Cette fonctionnalité s'effectue ainsi :

```
await contract.methods.getAllIds().call()
```

Les ids sont ainsi récupérés puis pour chaque identifiant, il faut récupérer l'article en question.
Cela se fait ainsi :

```
await contract.methods.articleContent(ids[i]).call()
```

Ils sont ensuite stockés dans une variable.

d. Récupération de l'historique d'un article

Pour récupérer l'historique d'un article, il faut d'abord récupérer le nombre d'historique qu'il possède grâce à

```
await contract.methods.getNumberHistorical(ids[i]).call();
```

Puis pour chaque numéro d'historique de 0 au nombre récupéré, il faut appeler cette fonction,

```
await contract.methods.getHistorical(ids[i], j).call();
```

Les historiques sont ainsi stockés dans un tableau et sont ensuite transmis à la vue.

III. Mise en forme au sein de React

La liste des articles ainsi que la liste des historiques des articles, sont stockées dans l'état global de l'application, cela permet d'y avoir accès à n'importe quel endroit de l'application.

a. Liste des articles

Afin d'afficher la liste des articles, après la récupération des articles, il faut modifier l'état global de l'application en utilisant la fonction dispatch de redux.

```
dispatch(getAllArticlesStore({ allarticles }));
```

La fonction getAllArticlesStore permet de modifier l'état des articles en transmettant les nouveaux articles à l'état.

```
const getAllArticlesStore = ({allarticles}) => ({
  type: ALL_ARTICLES,
  allarticles,
})
```

Figure : Fonction permettant de changer l'état

Cela fait ensuite appelle au reducer, qui renvoie un état avec les nouveaux articles.

```
case ALL_ARTICLES:
  const {allarticles} = action
  return { ...state, allarticles}
```

Figure : Permet de changer l'état des articles

Pour afficher les éléments dans le composant, nous faisons appel à useSelector qui permet de récupérer un élément d'un état.

```
const Lesarticles = useSelector(({ allarticles }) => allarticles)
```

b. Modification d'un article

Pour modifier, il faut d'abord récupérer son contenu, puis lorsque l'utilisateur veut modifier l'article, il clique sur submit. La modification est envoyée à la fonction modifyArticle qui communique avec le contrat.

La modification est alors prise en compte dans le contrat mais il faut ensuite mettre à jour l'état de l'application.
Ceci se fait grâce à la fonction getAllArticlesStore.

c. Affichage de l'historique d'un article

Pour mettre à jour les historiques des articles dans l'application, il suffit de mettre à jour l'état.
Cela se fait ainsi :

```
dispatch(updateHistorical({historical}))
```

d. Ajout d'un article

Lors de l'ajout d'un article, il faut mettre à jour la liste des articles au sein de l'application. Il suffit donc de faire appel à la fonction getAllArticlesStore.

IV. Intégration de Fortmatic

Pour intégrer la Dapp Fortmatic, il faut désactiver Metatask.
Il faut ensuite ajouter le nœud personnalisé de ganache:

```
const customNodeOptions = {
  rpcUrl: 'http://127.0.0.1:7545', // your own node url
  chainId: 0x539 // chainId of your own node
}
```

Puis il faut créer le provider :

```
let fm = new Fortmatic('pk_test_DA712556385A2857', customNodeOptions);
window.web3 = new Web3(fm.getProvider())
```

Et récupérer le compte utilisateur pour intéragir avec le contrat.

```
const [account]= await window.web3.eth.getAccounts();
```

Grâce à l'ajout de ces quelques lignes, Fortmatic est intégré.