Create ReactJs application :

J’ai utilisé la commande npm pour installer les modules nécessaires à la création d’une application ReactJS :



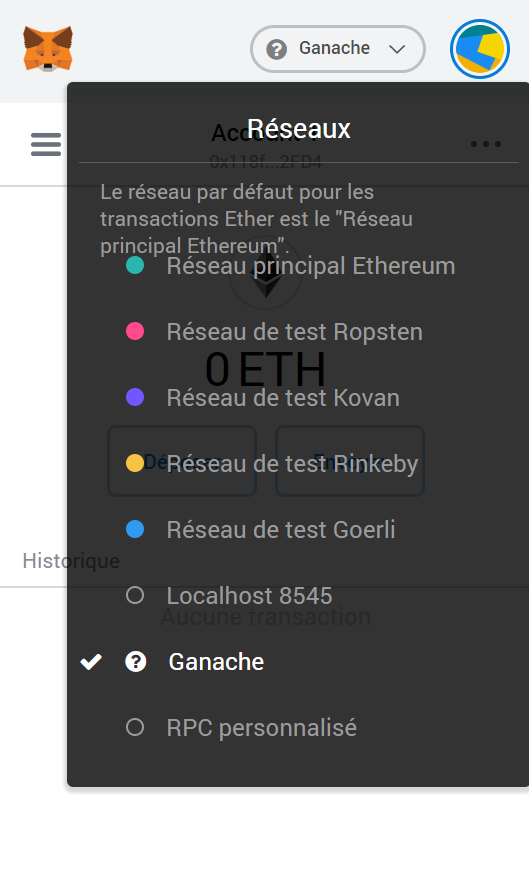
Pour créer une nouvelle application :



Pour lancer l’application ReactJS :



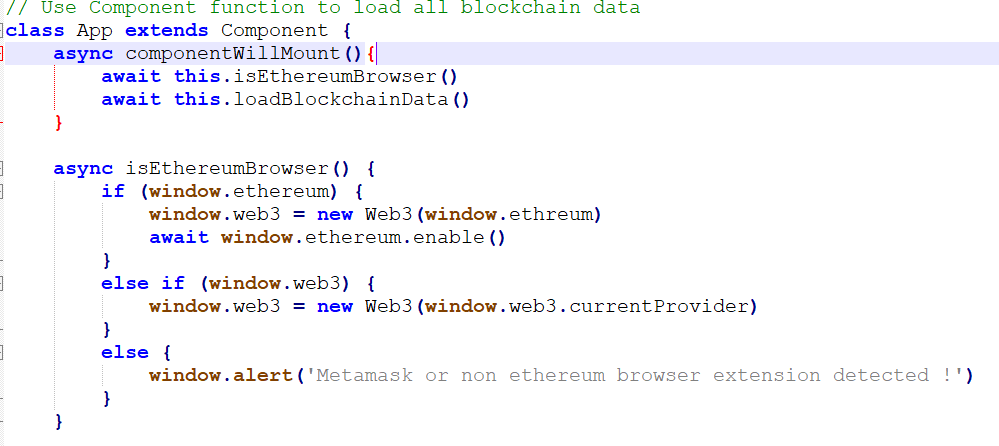
Ensuite, il faut configurer Metamask pour se connecter à Ganache.



On utilise web3 pour interagir avec Ethereum.

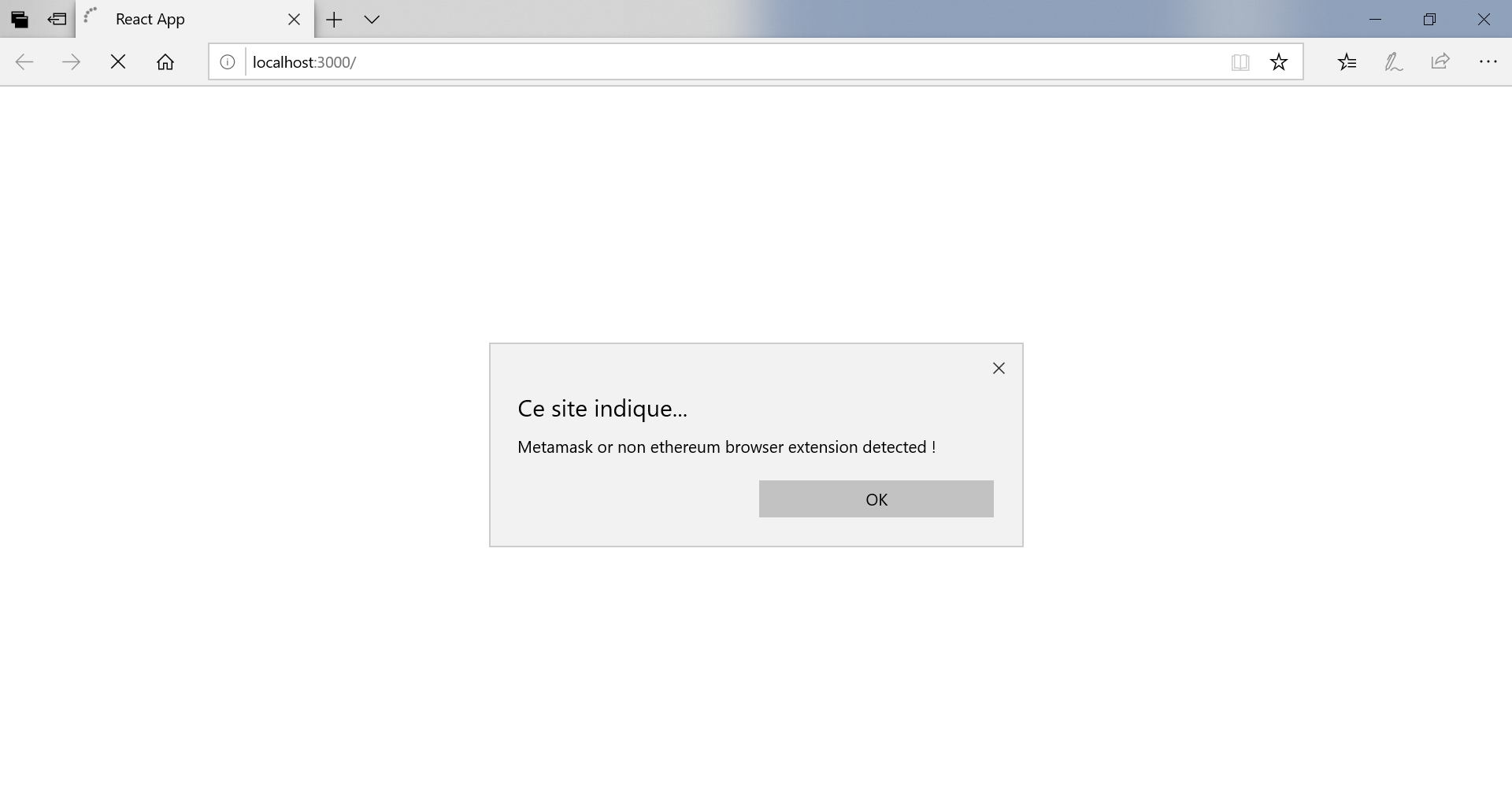


Ensuite, lorsqu’on se connecte à son application ReactJs, on regarde si une extension Ethereum est présente (exemple Metamask), si ce n’est pas le cas on affiche le message d’erreur avec « alert ».

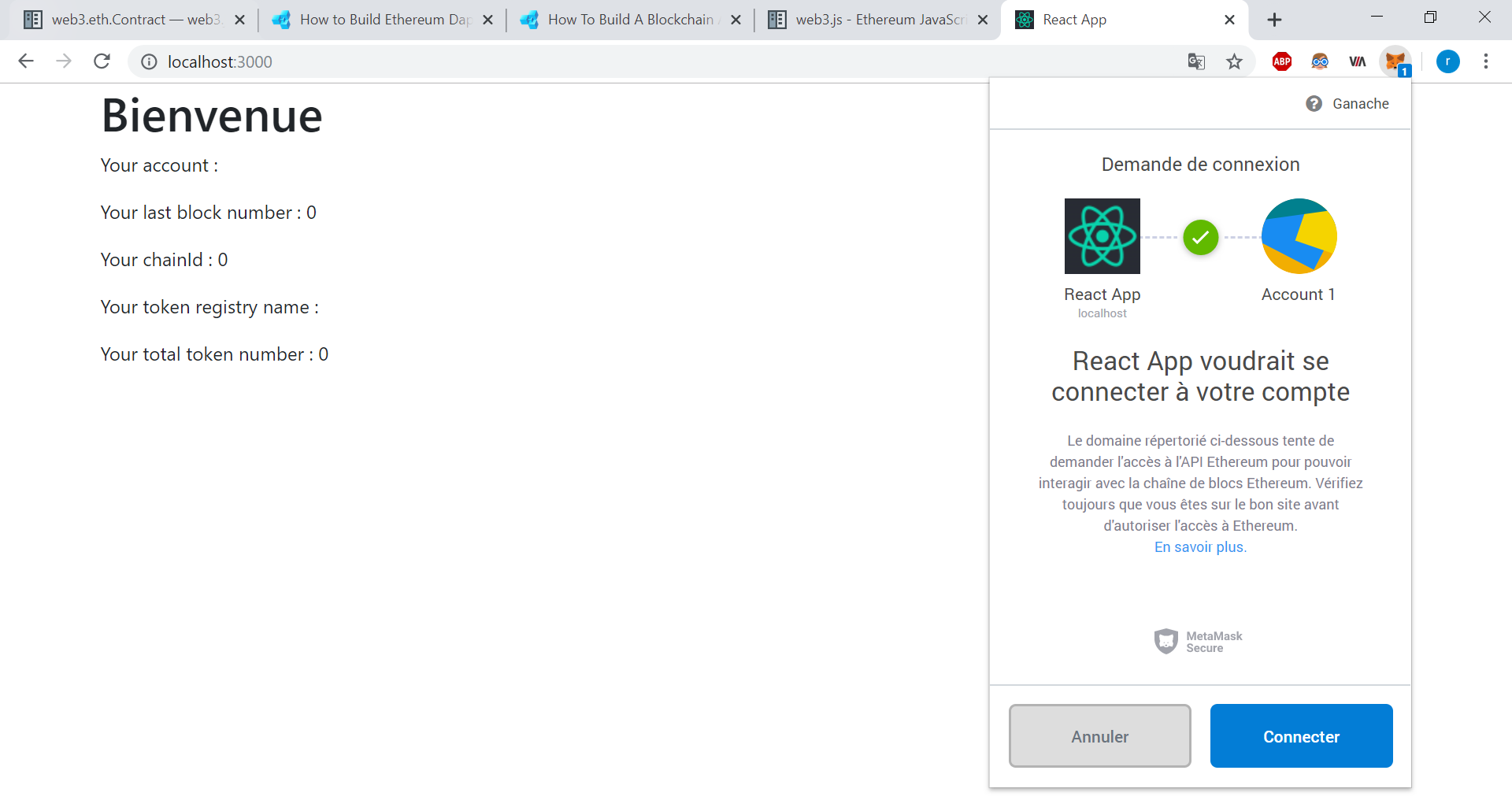


**Résultat :**

1. Internet Explorer n’a pas de module Ethereum présent donc message d’erreur :



1. Chrome a un module Ethereum, donc ReactJs demande à Metamask s’il peut accéder à ses informations.



Ensuite, nous déployons le contrat de l’application and obtenons les informations du contrat à savoir « chainId », « last\_block », « nameToken », « totalTokenNumber », « account address value ». Pour cela, on récupère toutes les adresses de Ganache avec « web3.eth.getAccounts() » puis on récupère la première adresse. Ensuite, pour le chainId et le numéro du dernier block, on utilise les fonctions fournie par web3 : « getChainId() » et « getBlockNumer() ».

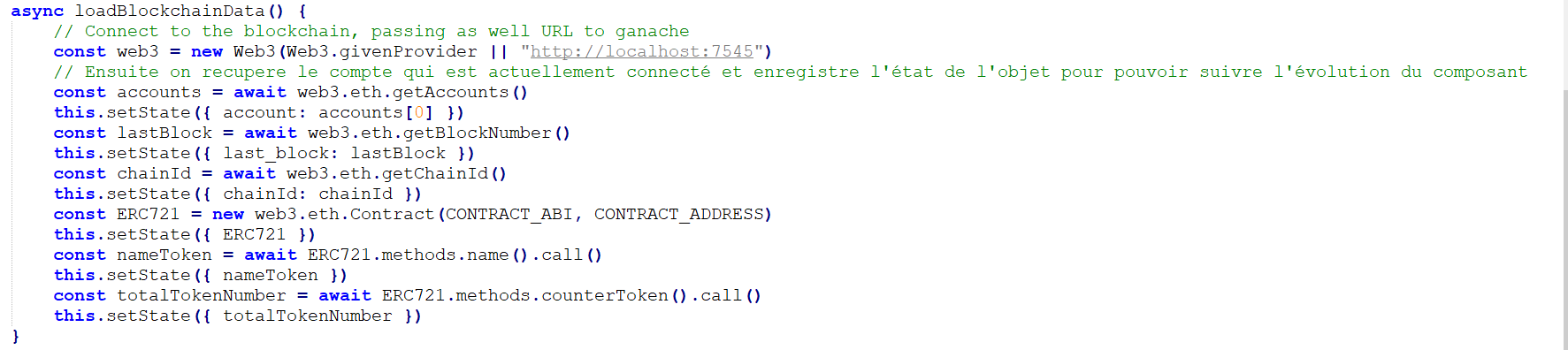
Concernant le nom du token et le nombre total de token, nous devons d’abord déployer le contrat sur l’application. Pour cela on utilise la fonction « .Contract (abi, address) » qui prend en paramètre l’ABI du contrat à savoir celui présent dans ERC721.json et l’adresse du contrat ERC721 que l’on peut obtenir en faisant :



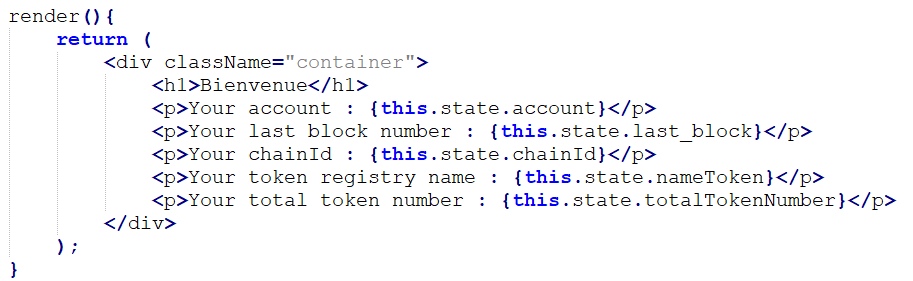
L’ABI et l’adresse du contrat ont été ajoutés dans un fichier externe « config » que l’on importe dans le code de l’application ReactJs « App.js ».



Après, cela nous renvoie un objet qui contient toutes les méthodes et variables. Pour cela, on utilise « objet\_contrat.methods.nom\_Methode().call() » pour récupérer la valeur d’une variable.



Ensuite, on doit pouvoir afficher et mettre en forme les valeurs à afficher dans l’application. On le fait grâce à « render() »



**Résultat :**

