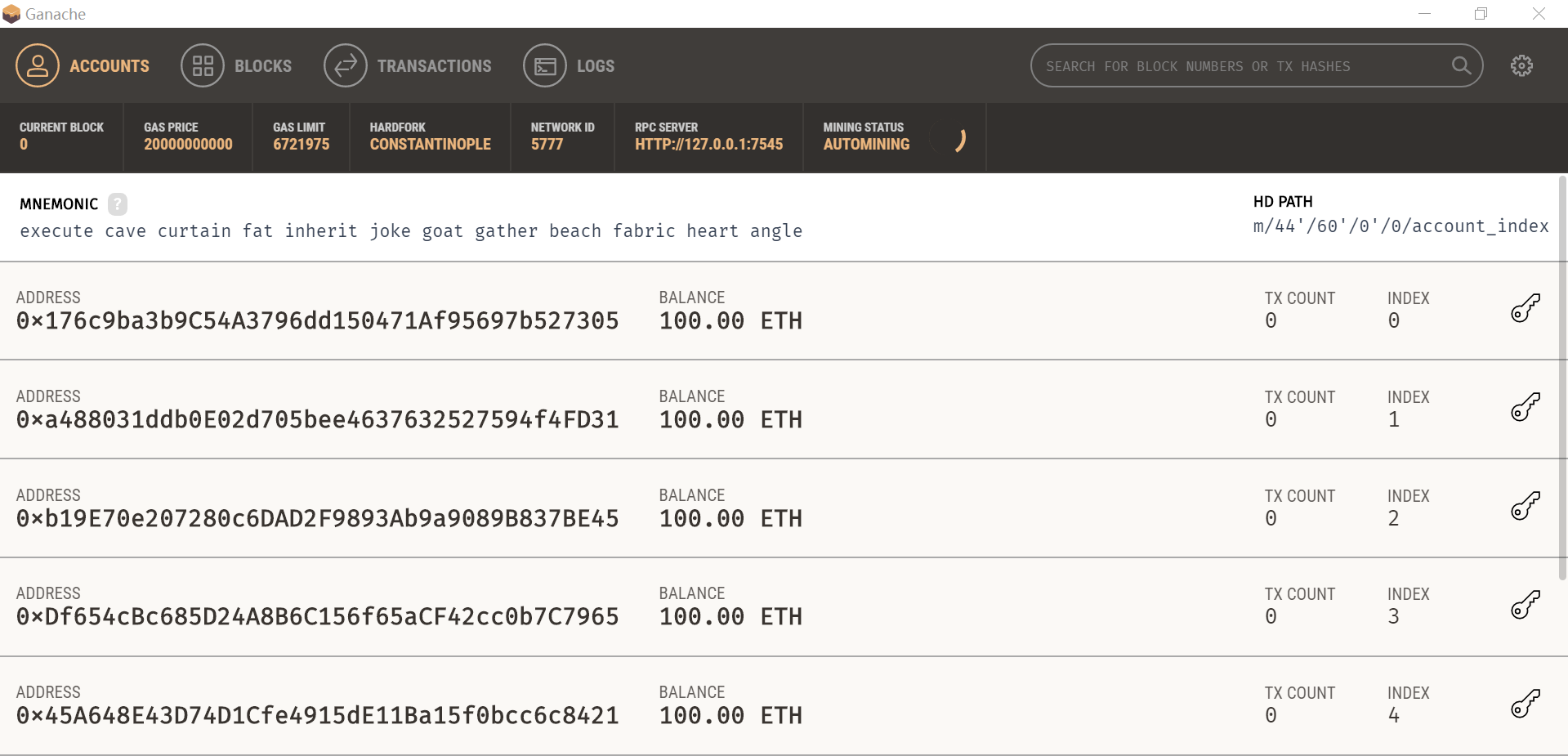
How to build a Dapp

1. Setting up Ethereum
   1. Introduction to Ethereum

Ethereum is an open-source software platform based on blockchain technology which allows developers to build and deploy decentralized applications. Like Bitcoin, Ethereum is also a distributed public blockchain network. Yet, they have different purposes and capabilities. Bitcoin offers peer-to-peer e-cash system that enables online payments, while Ethereum targets at running codes for any decentralized applications deployed on it. On the Ethereum, Ether is the cryptocurrency that fuels the network, which is also used for paying code execution. Smart contract is a "phrase" used to describe an Ethereum application that can facilitate the exchange of digital assets. It is self-operating computer program and will run exactly as coded without any possibility of downtime, censorship, fraud or third-party interference. Ethereum provides a decentralized virtual machine called Ethereum Virtual Machine (EVM), a runtime environment for smart contracts.

* 1. Ganache installation

Ganache is a simulated and personal blockchain for Ethereum development you can use to deploy contracts, develop your applications, and run tests. You can download Ganache from the link: <https://truffleframework.com/ganache> and install it:



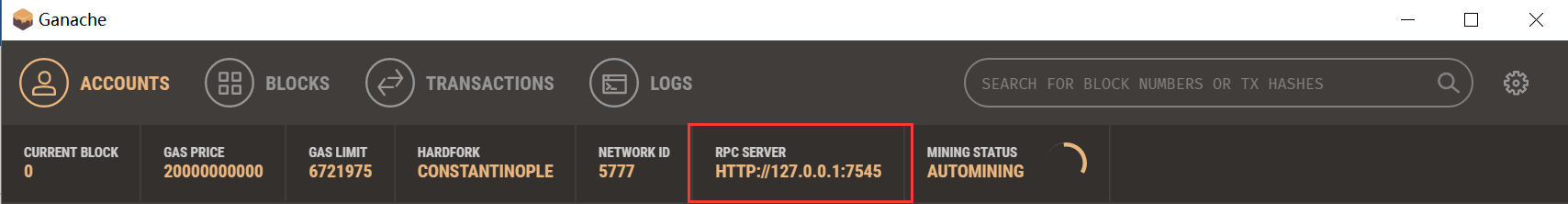
As shown in the above figure, Ganache has booted, and you have a local blockchain running. It provides us 10 accounts with addresses on the local Ethereum, and each account has 100 fake 100 ethers (these ethers are not worth anything on the main Ethereum network).

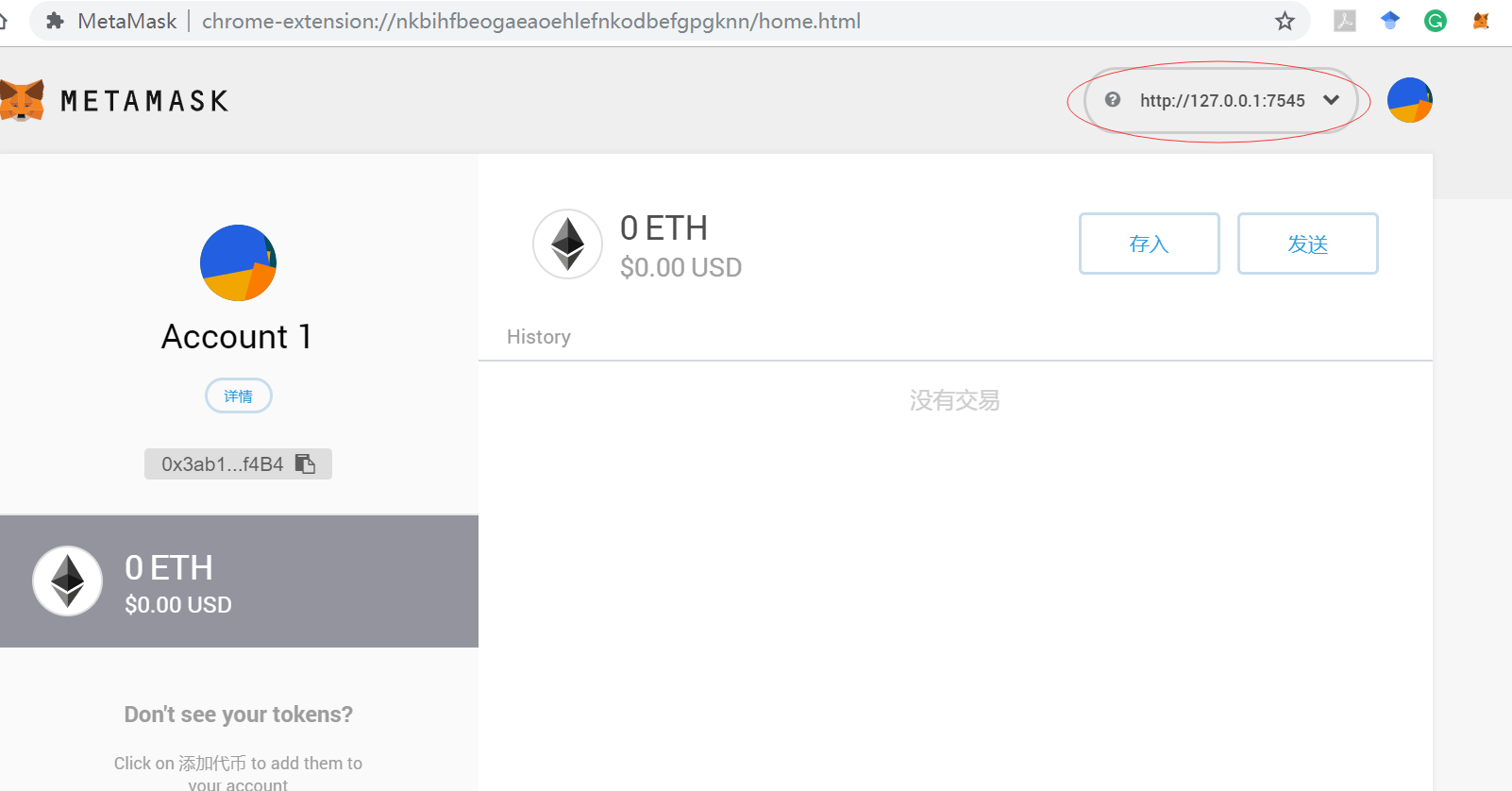
* 1. Using Metamask to connect to Ganache

The first step is to install the Google Chrome browser, and in order to connect to Ganache (note that the blockchain is a network), a Metamask extension (plugin) for Chrome should be installed from the link:

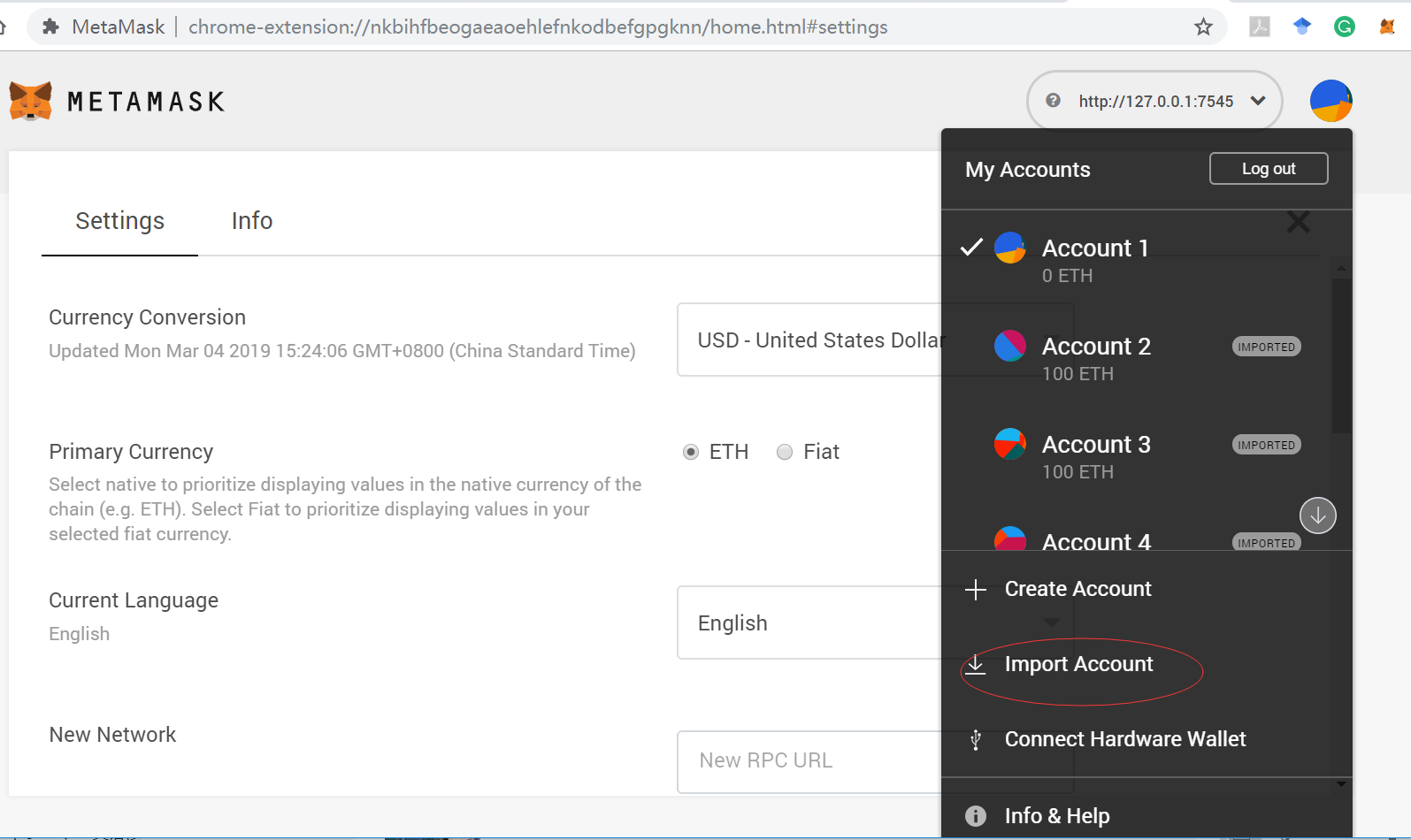
<https://chrome.google.com/webstore/detail/metamask/nkbihfbeogaeaoehlefnkodbefgpgknn?hl=en>.

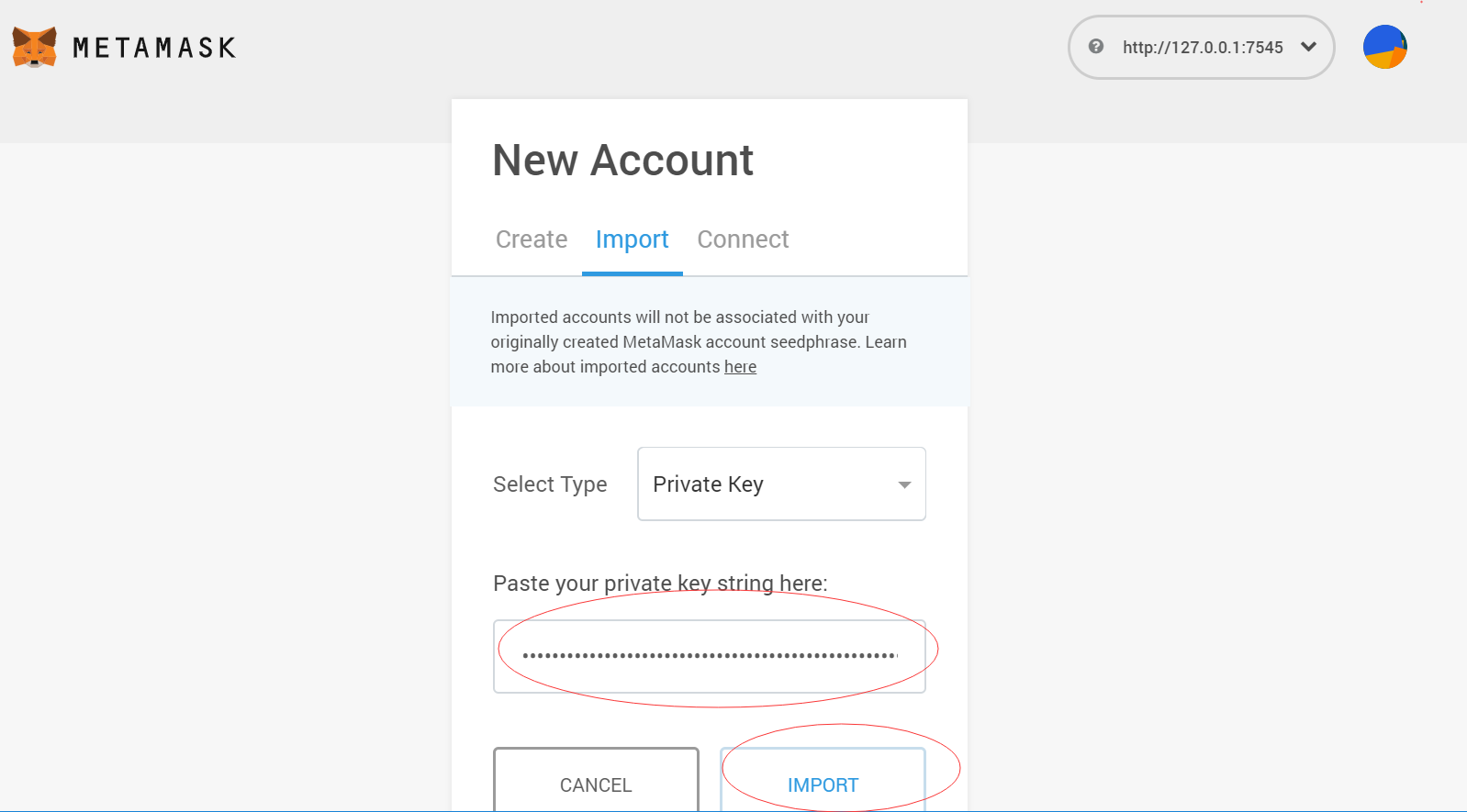
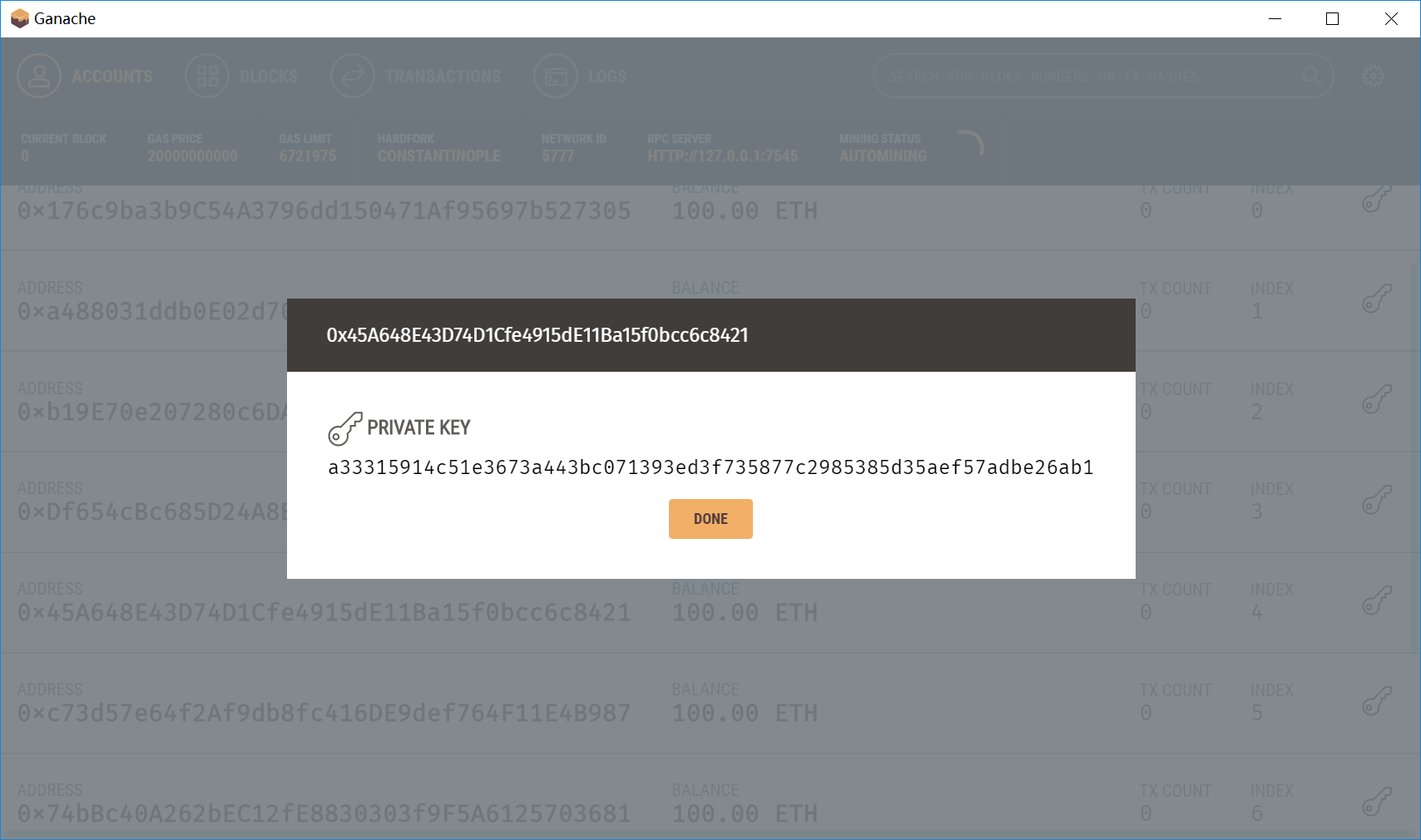
MetaMask allows us to create and manage our identities via private keys, local client wallet, and hardware wallets. After registering and entering the RPC server IP address (of Ganache) in Metamask, you can connect to Ganache via Metamask:



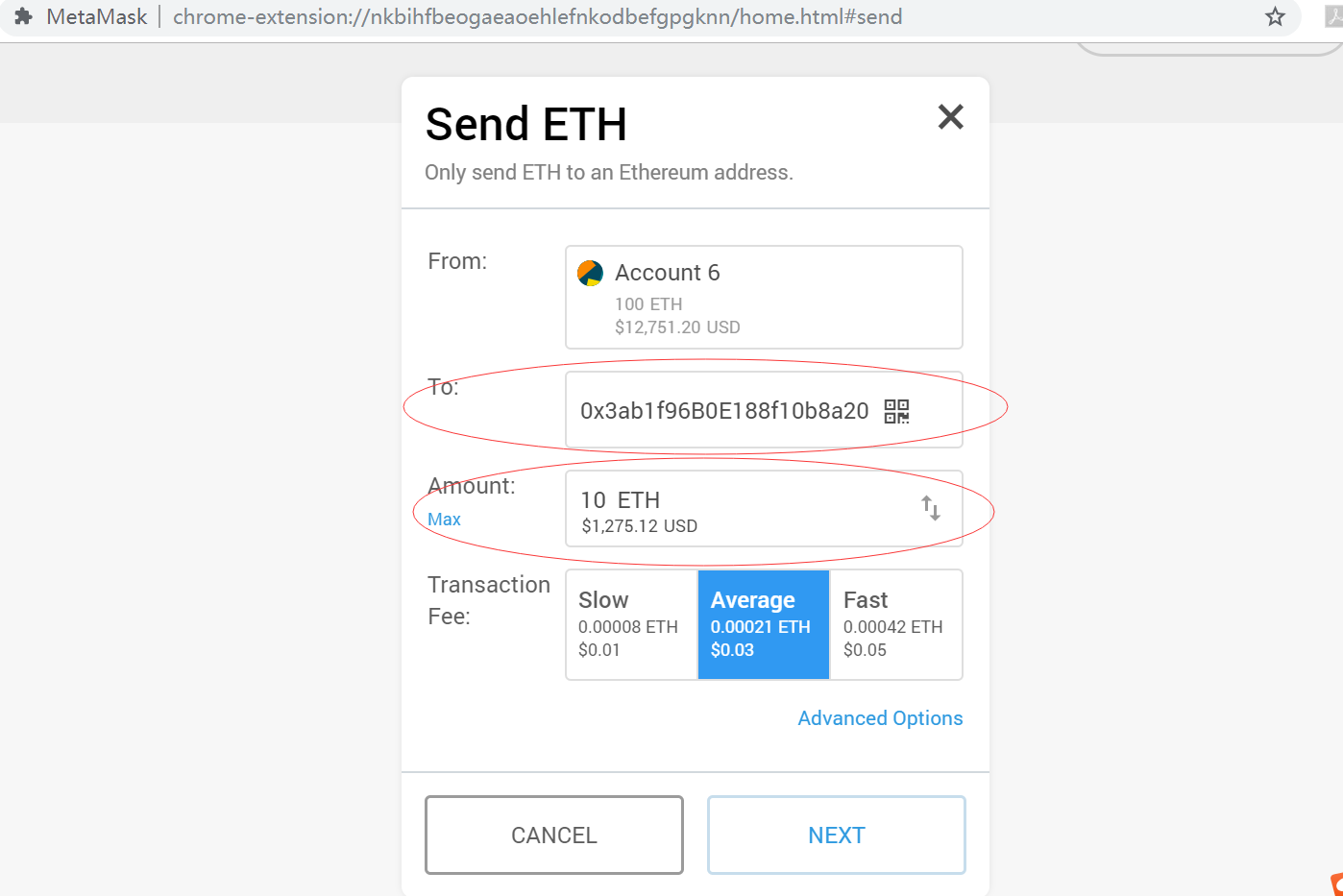


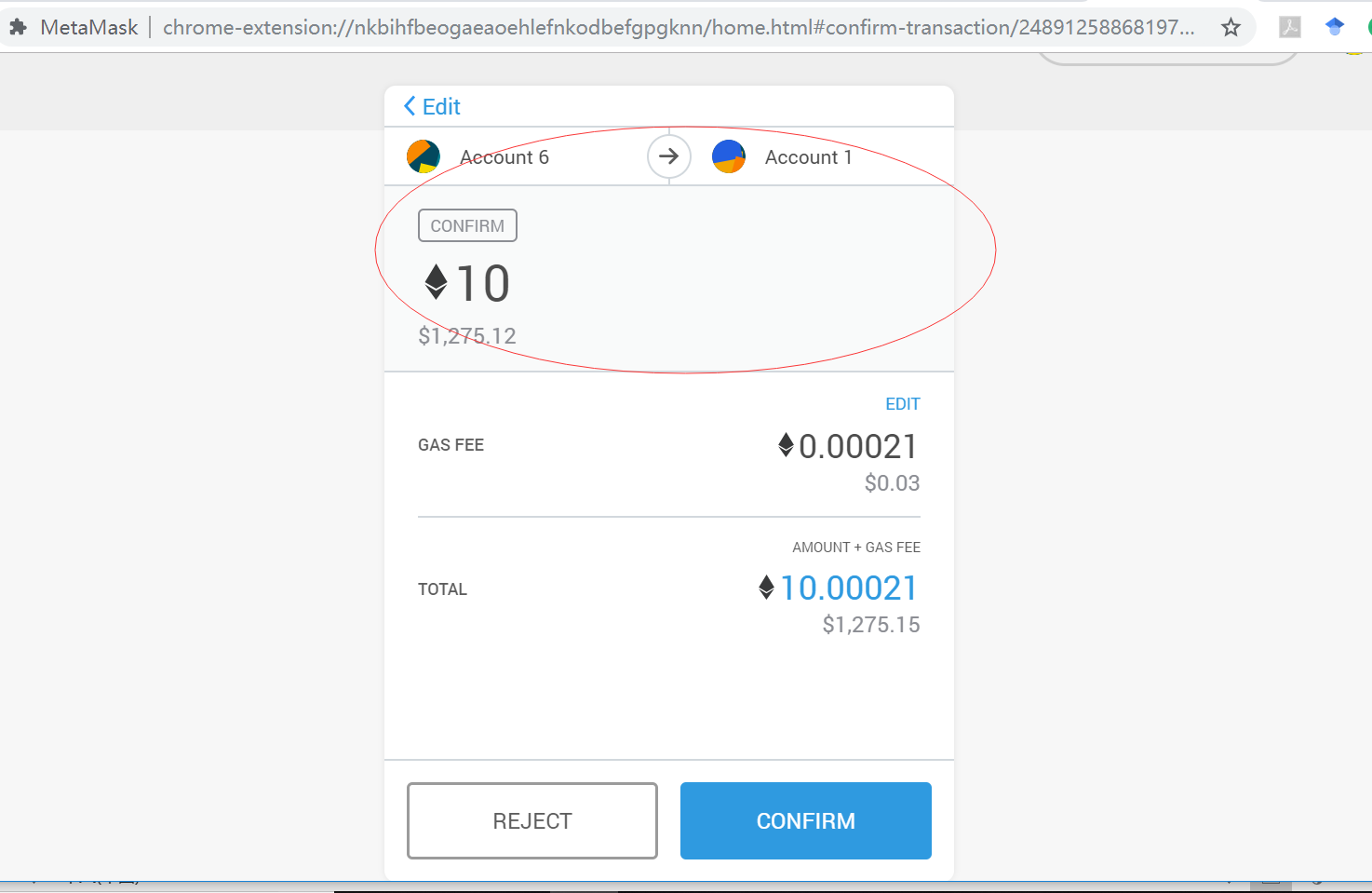
Then, you can import the accounts (provided in Ganache) by copying private keys in Ganache and pasting them to Metamask: (Note that Account 1 is the original account provided by Metamask, Account 2 to 5 are accounts imported before writing this tutorial)



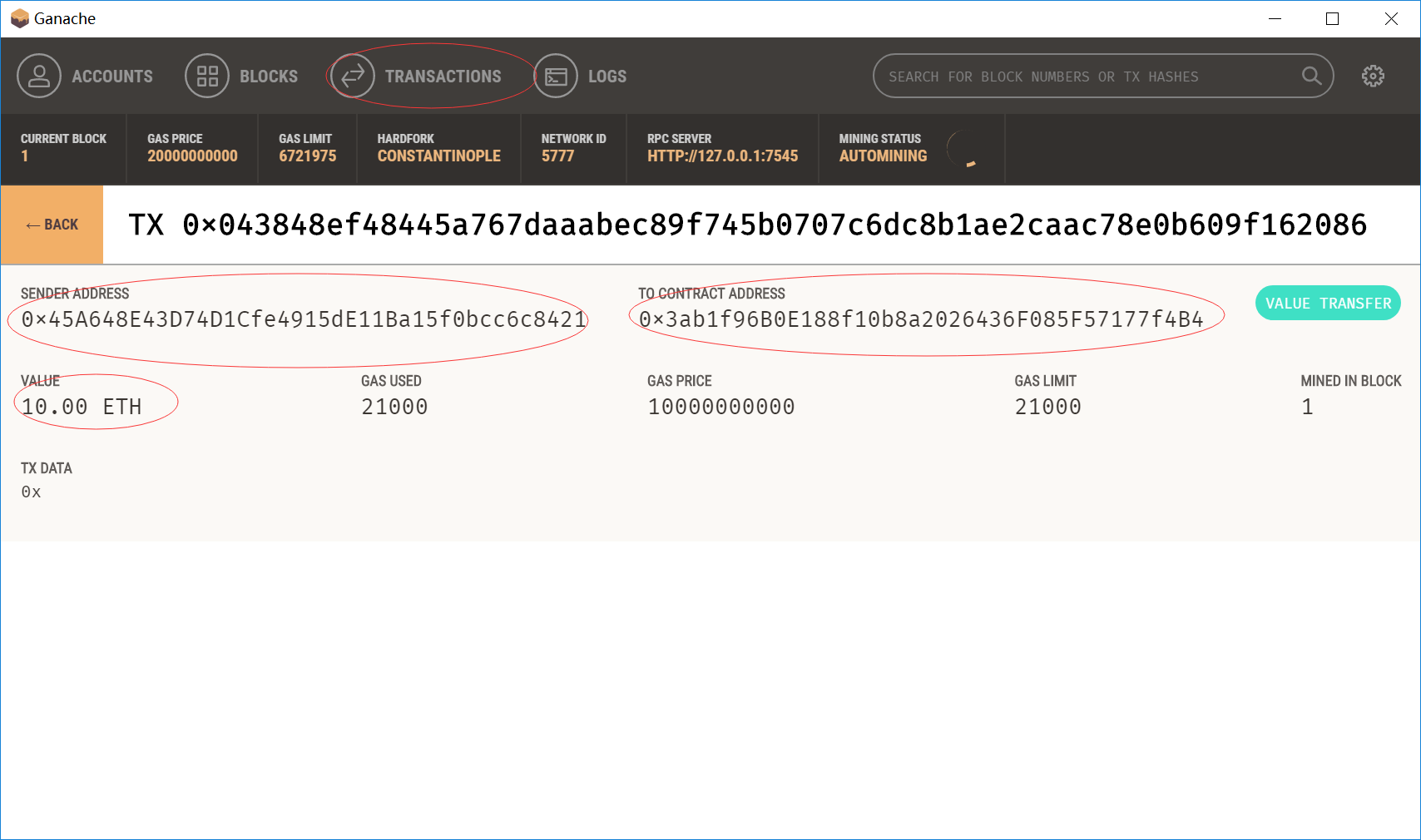


Afterwards, you can transfer 10 ethers from Account 6 (which has 100 ethers) that you just imported to Account 1 (which has 0 ether):

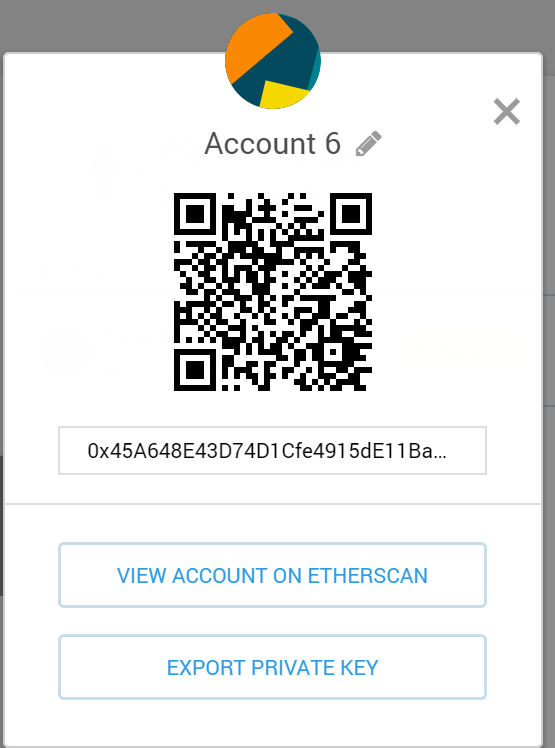
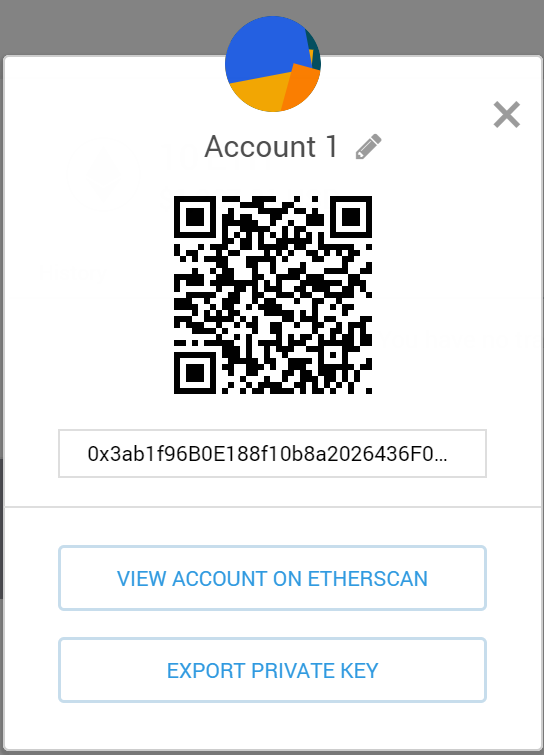




Also, you can track this transaction information through the Ganache:



where the addresses of Account 6 and Account 1 are listed below:

Acknowledgement of the following links:

<https://truffleframework.com/ganache>

<http://www.dappuniversity.com/articles/the-ultimate-ethereum-dapp-tutorial>

<https://medium.com/edgefund/ethereum-development-on-windows-part-1-da260f6a6c06>

<https://en.wikipedia.org/wiki/Ethereum>

1. Setting up development environment on Windows 10
   1. Introduction to Smart contract

Smart contract is a computer program executed in a secure environment that directly controls digital currencies or assets. It is in charge of reading and writing data to blockchain, as well as executing business logic, for example, determining whether an asset should go to a receiver or should be returned to a sender from whom the asset originated.

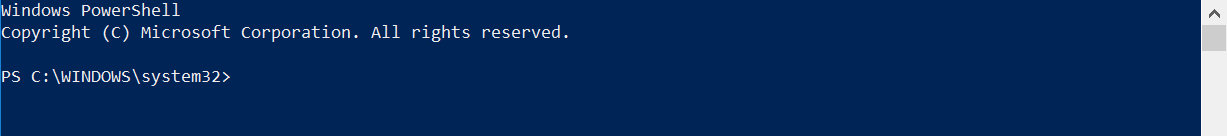
* 1. The Solidity programming language

Solidity is a contract-oriented, high-level language for implementing smart contracts. It was influenced by C++, Python and JavaScript and is designed to target the EVM as mentioned above. It is statically typed, supports inheritance, libraries and complex user-defined types among other features. For example, most of the control statements from JavaScript are available in solidity except for 'Switch' and 'goto'. For specifications, you can refer to the link: https://solidity.readthedocs.io/en/v0.5.4/.

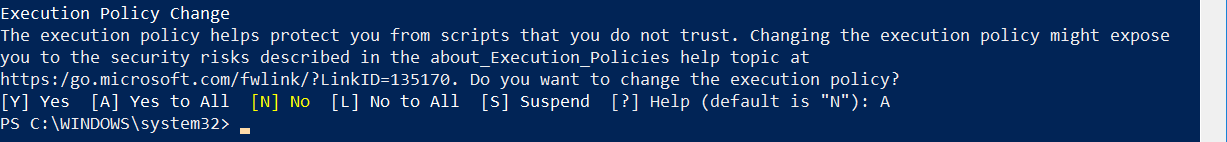
* 1. Chocolatey installation

We aim at developing a Dapp on Windows 10, so the first step is to install Chocolatey via PowerShell, which is a package manager for Windows.

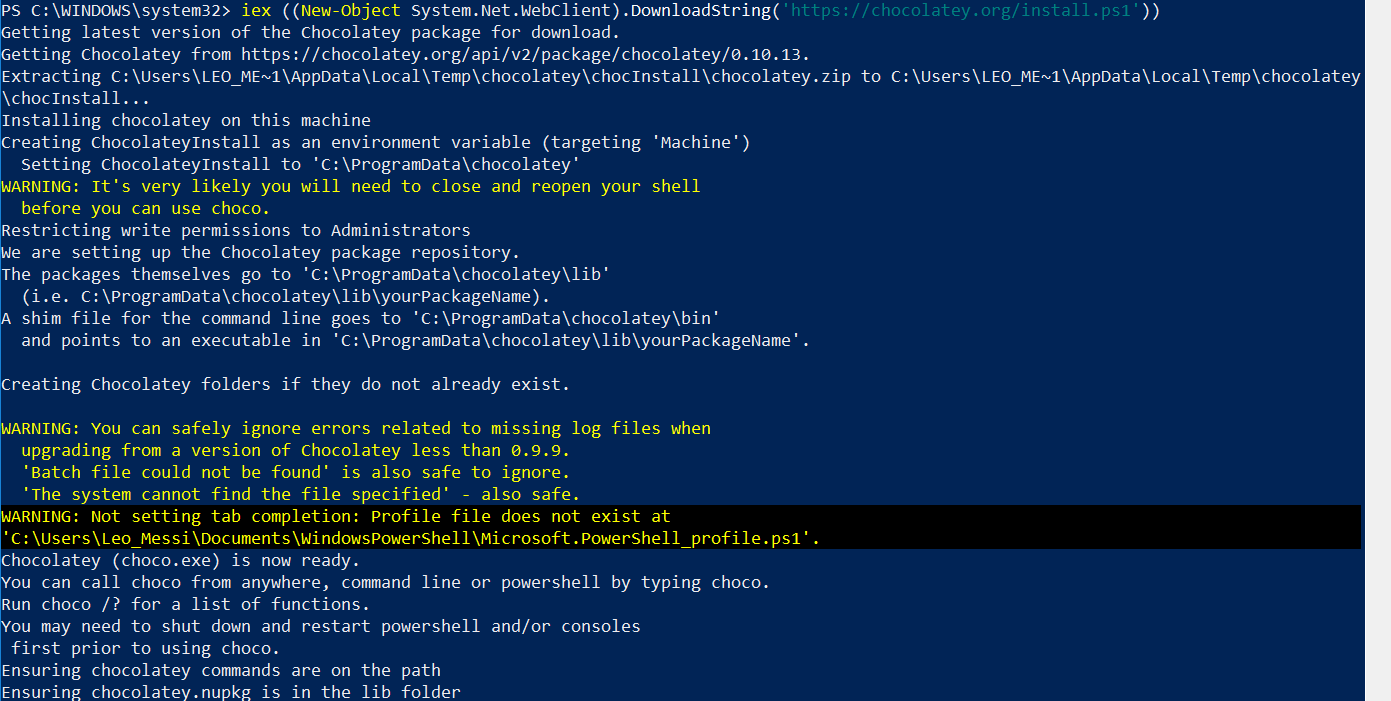
Start by opening a PowerShell window with administrator rights:



In PowerShell, type 'Set-ExecutionPolicy Bypass' and then 'A' to accept elevated privileges:

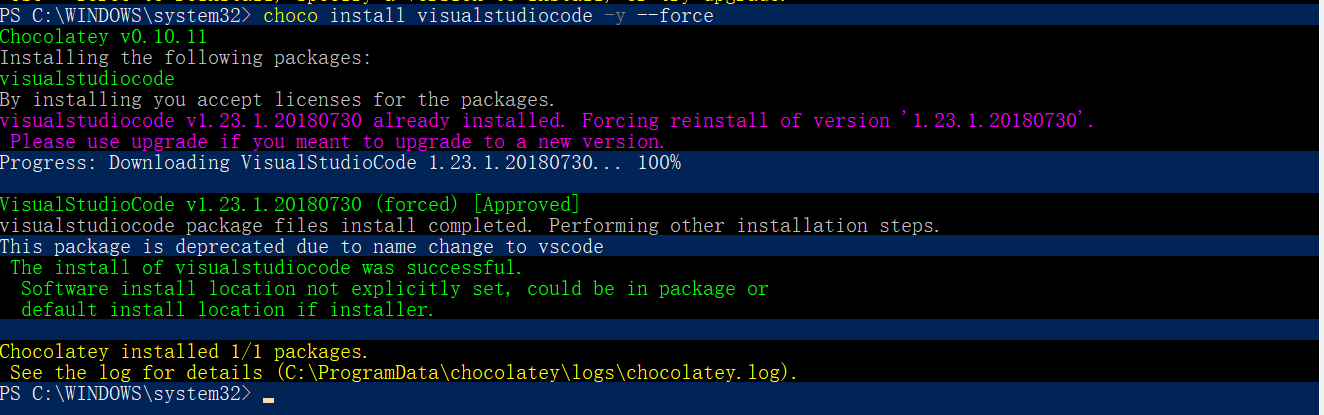


Then, enter the following for installation "iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))":

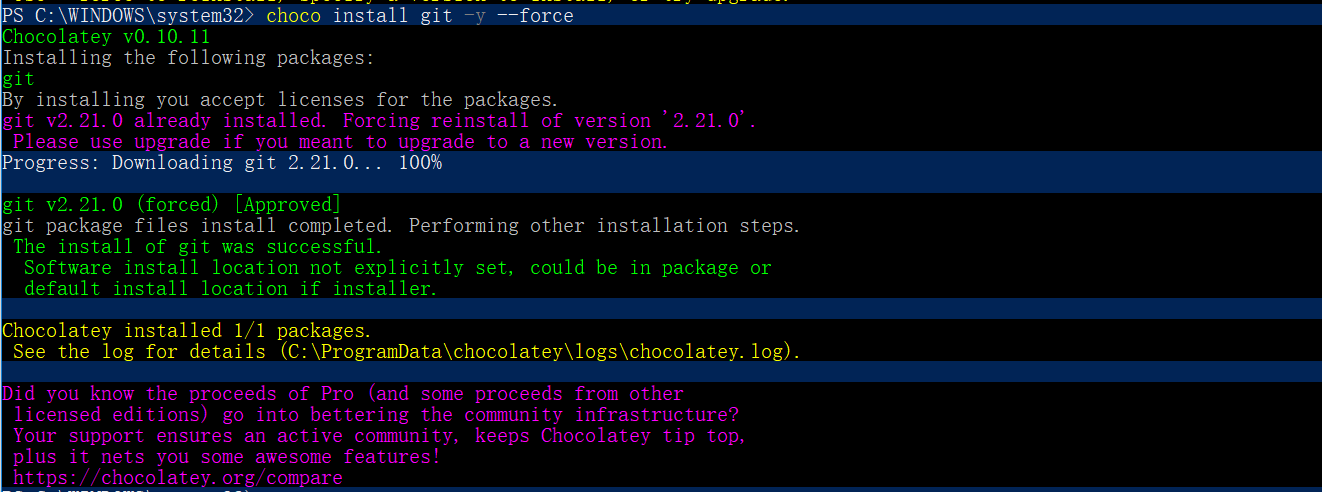


* 1. Other (required) tools installation

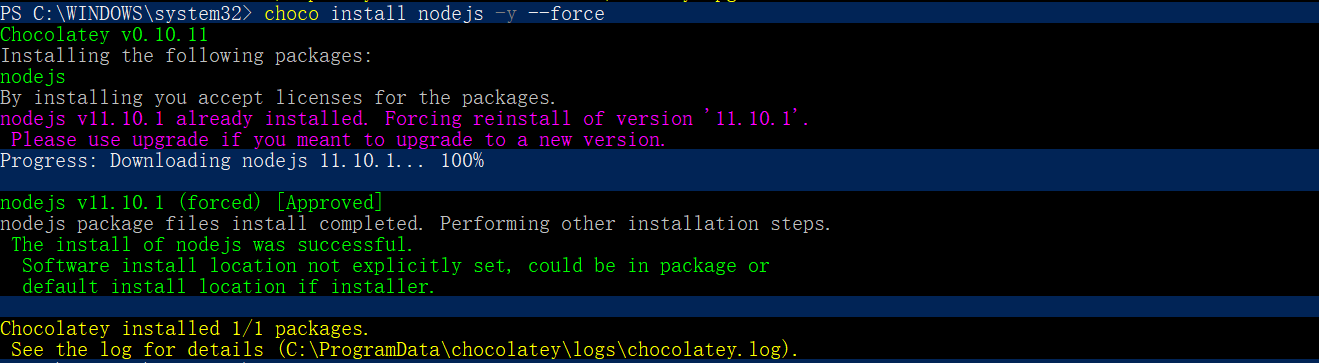
Reopen PowerShell run as administrator, and install the IDE (Visual Studio Code) by entering "choco install visualstudiocode -y": ("--force" is used for re-installation)



Then, install the source control framework (Git) by entering "choco install git -y":



Finally, install the Node.js, a cross-platform JavaScript runtime environment, including the node package manager (NPM) by entering "choco install nodejs -y":



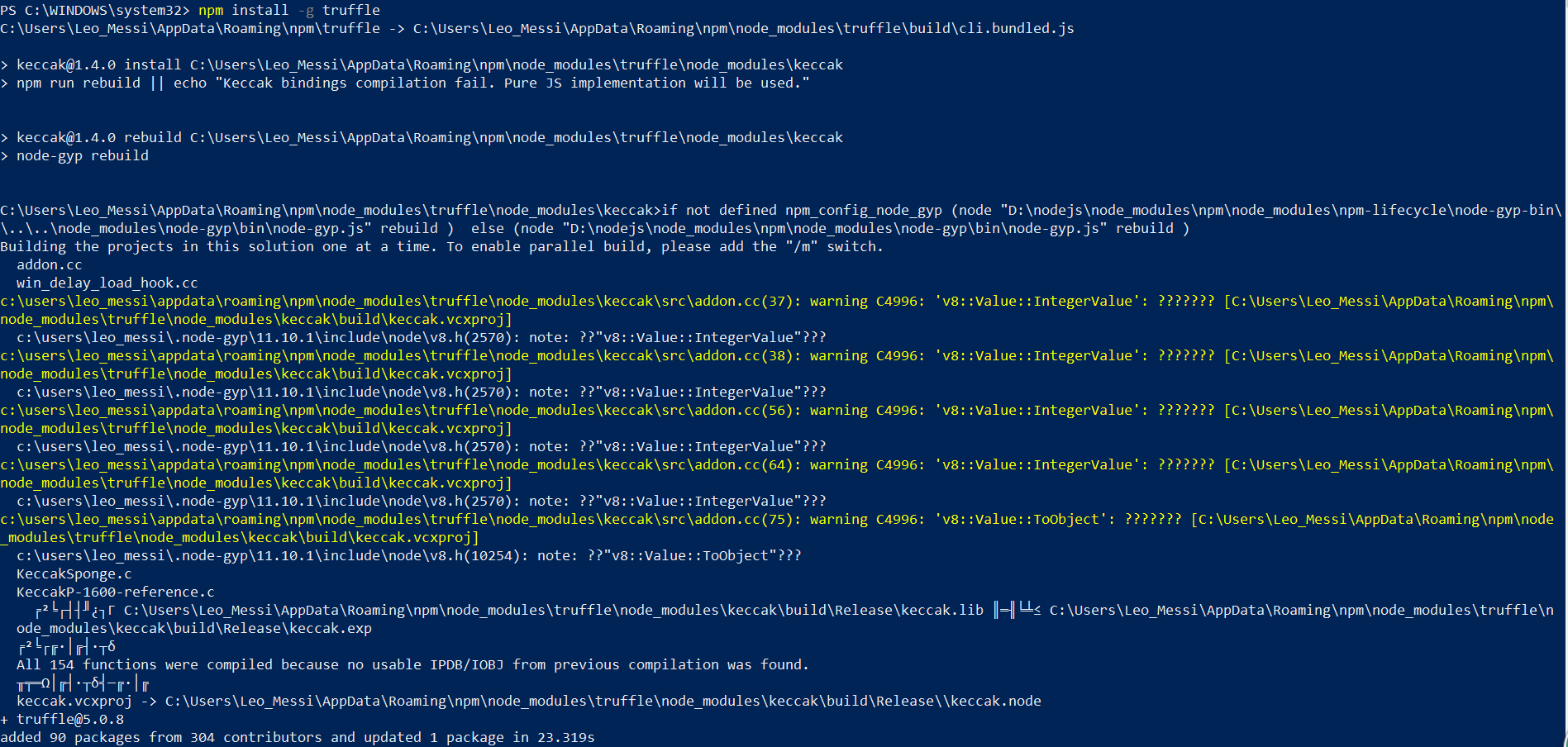
Upon installation, you can check successful installation of these tools by typing "node -v" and "npm -v" (i.e., their versions):



* 1. Truffle framework

Truffle framework allows us to build our Dapp on the Ethereum blockchain, which provides a suite of tools allowing us to write smart contacts with the Solidity and also enables us to test our smart contracts and deploy them to the blockchain.

With Node.js installed, use NPM to install the truffle framework by entering "npm install -g truffle":



You can verify the successful installation by checking its version via "truffle -v":



At this stage, all of the required packages are installed, then you can develop Dapp via the installed IDE of VS Code.

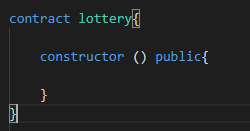
1. Write Your First Smart Contract

In this section, you will build a smart contract of a lottery system. We want to build a lottery application in this tutorial so we need to build a contract that can 1. store the prizes in the lottery pool (prizes function below that map from a index to a prize structure), 2. receive money from a user, select a prize randomly, and return the corresponding money back to the user (draw function below).

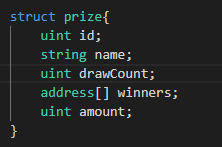
First of all, you create a lottery.sol file in contract folder.

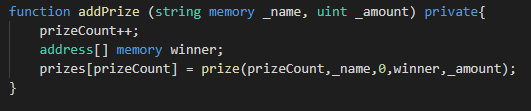


Then you create a contract named lottery with its constructor.

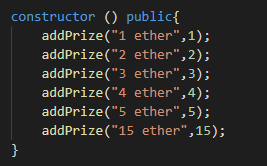


You define a struct to store the prize for the lottery, a mapping function to get the prize, and a function to edit the prize.

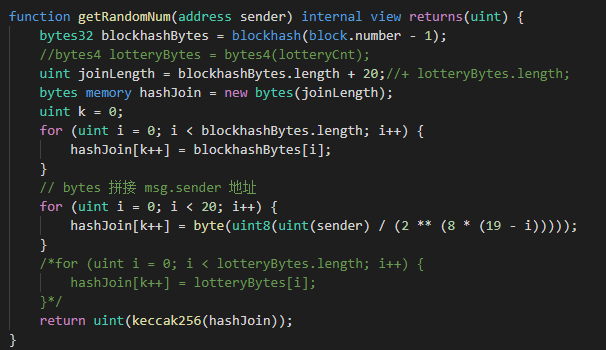




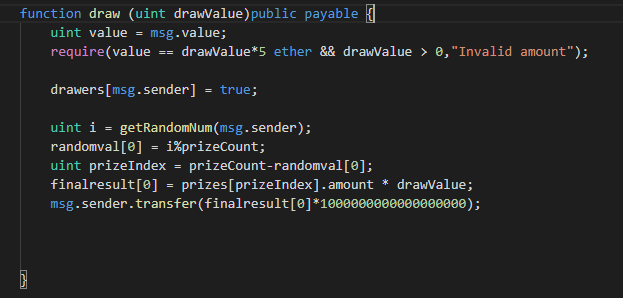
Then in your constructor function, you just add some prizes to the lottery pool, in this example, we add 6 prizes.



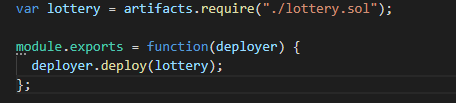
To make the lottery fair, you must be able to generate random values, you can define the function as follows.



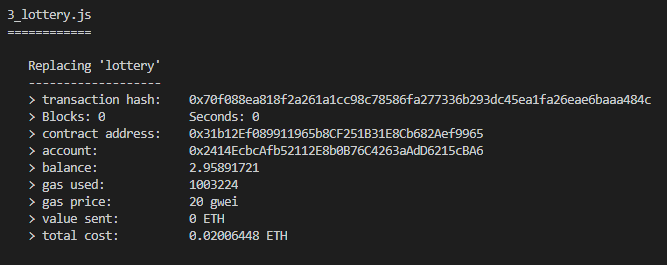
Then you define the draw function for a user to draw, this function requires an amount of ether and returns an amount of ether back to the user according to the generated random value.



Finally, you create a js file in the migrations folder for migrating the lottery.sol and then run truffle migrate in the shell.



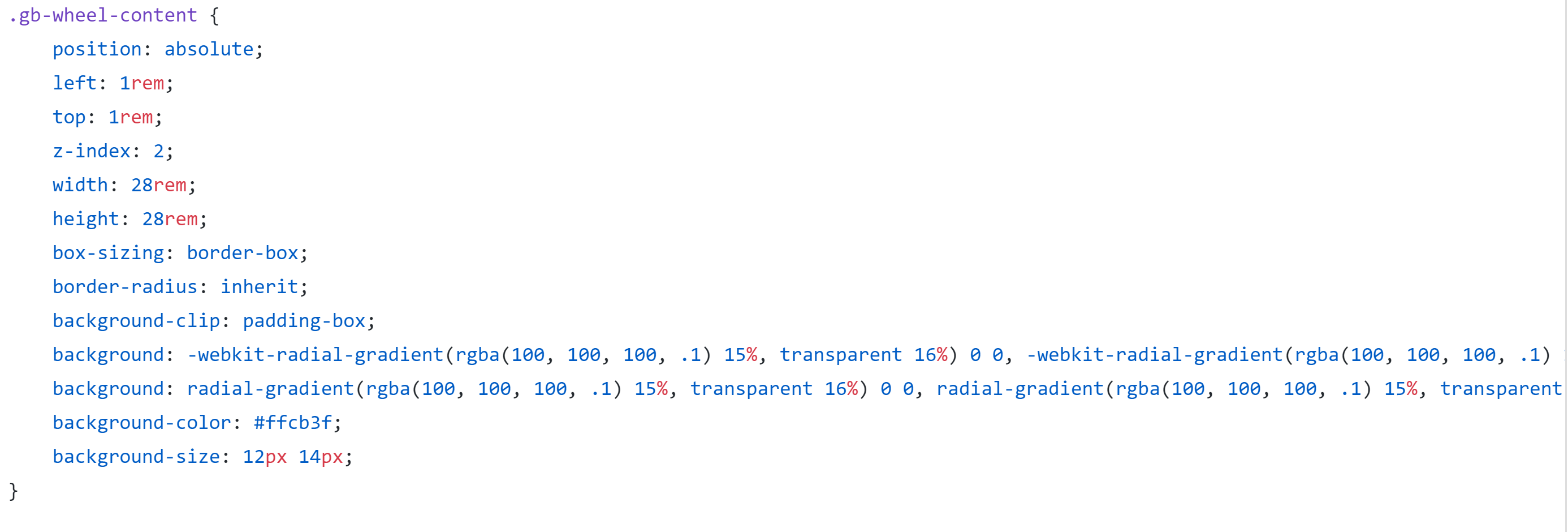




1. Prepare the Frontend for your Dapp

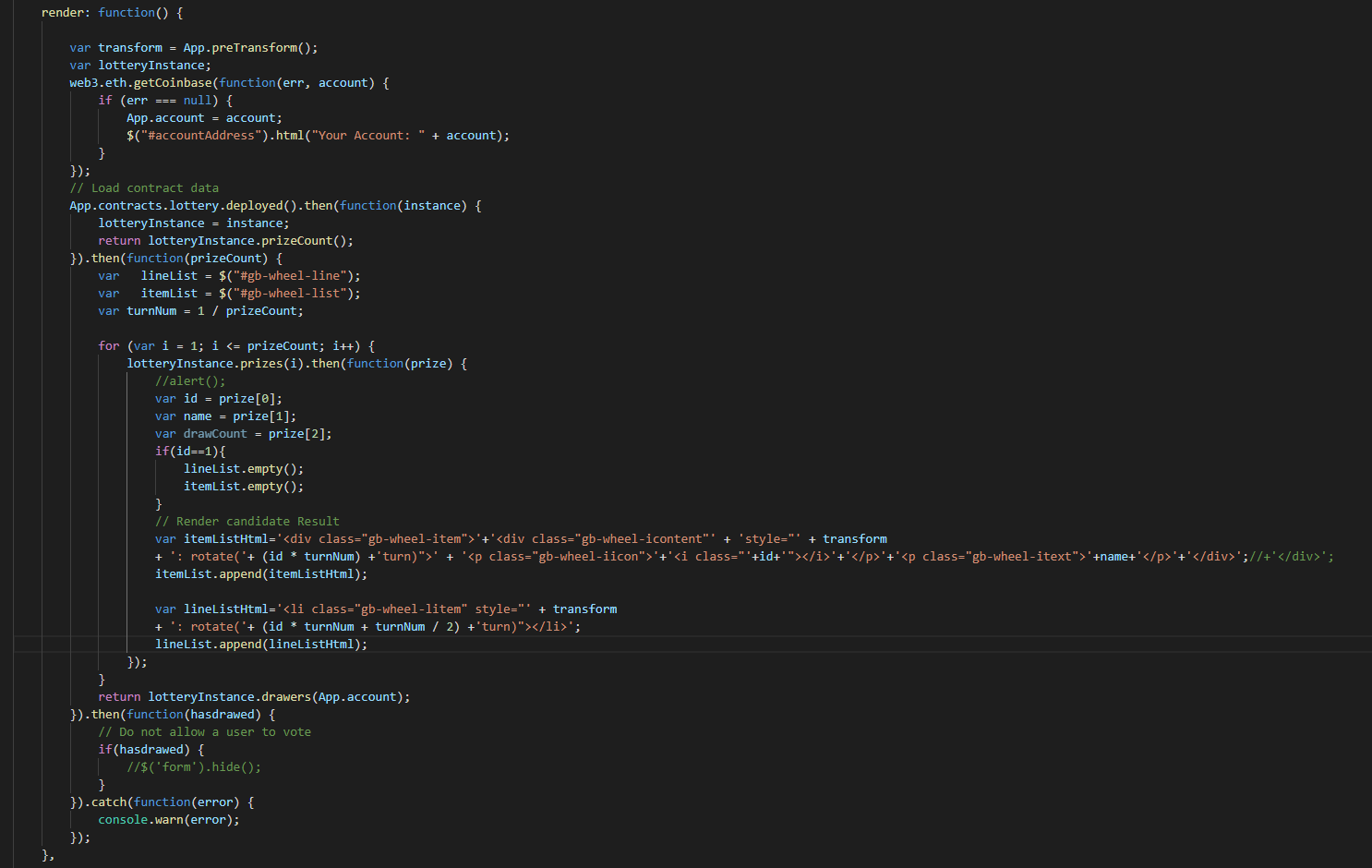
In this section, you are going to learn to build a website for using your Dapp. First of all, we have a html file and a ccs file for a simple website. Here in the html file, the element with id gb-wheel-line denotes the lines in the wheel and the element with id gb-wheel-list denotes the prizes. Items will be added to the two elements by a js file which we give below. The method we implement to rotate the wheel is in the css file, once the button is clicked, the position of the prizes and the lines will be changed to perform rotation.





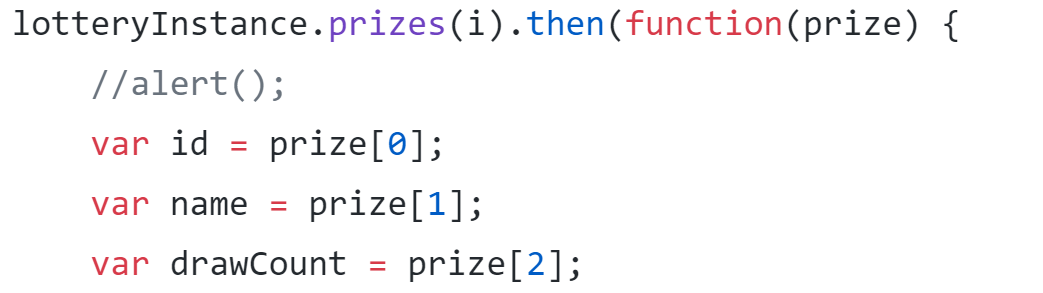


After that, we need to build a js file to interact with your smart contract. In your js file, you have a render function to fetch prizes from the smart contract and print the prizes out in the website, and a draw function that pays the smart contract for draw.





The main interactions between the js file and the backend in the two functions are as follows



The render function generates a lottery instance and get prizes from the backend using the prizes mapping defined in the lottery.sol file.



The draw function generates a lottery instance and pass the user’s account together with the user’s draw value to the backend using the draw function defined in the lottery.sol file.

Acknowledgement of the following links:

<https://solidity.readthedocs.io/en/v0.4.21/>

<https://en.wikipedia.org/wiki/Smart_contract>

<https://searchcompliance.techtarget.com/definition/smart-contract>

<https://truffleframework.com/>

<https://medium.com/edgefund/ethereum-development-on-windows-part-1-da260f6a6c06>