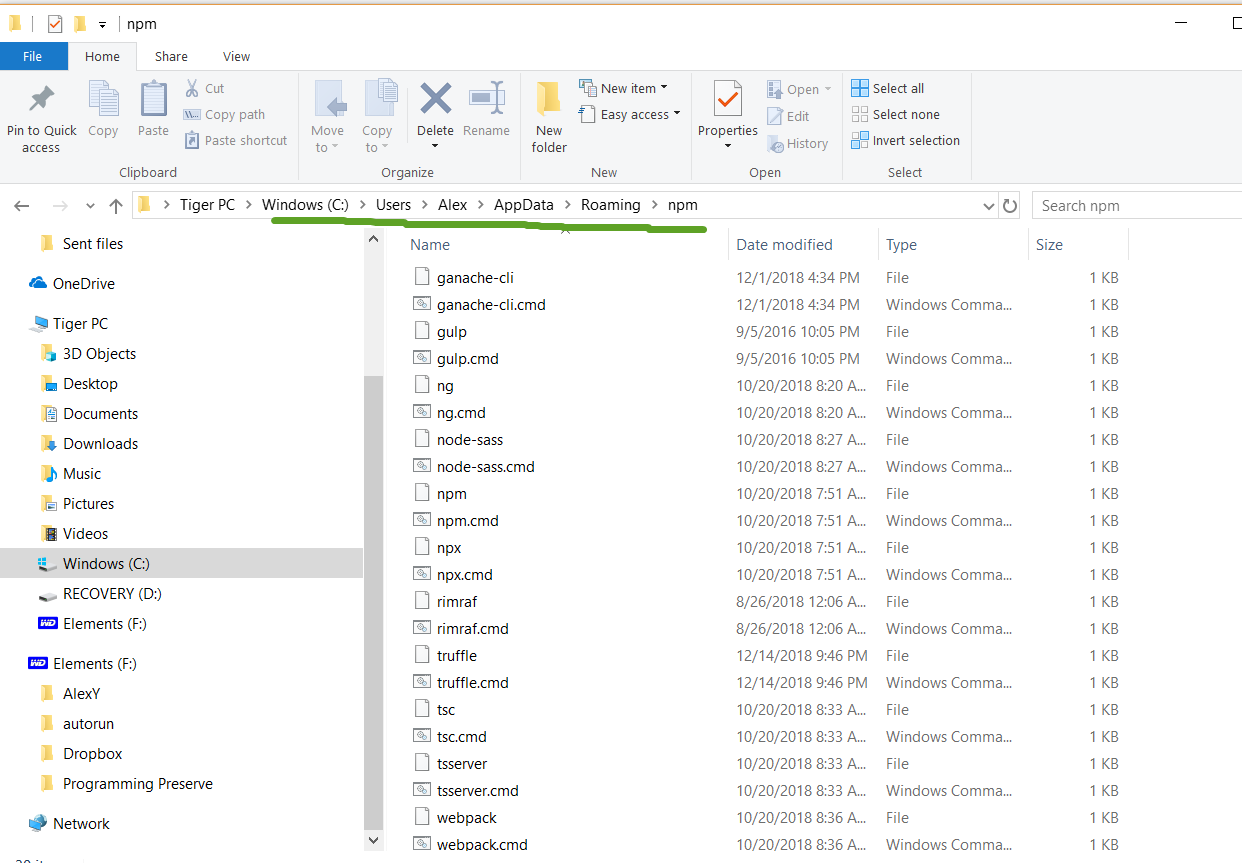
### Architecture Notes to the Flea Market on Ethereum Blockchain

Based on <https://medium.com/b2expand/install-web3-fc96b885a7b> to be able successfully install web3 the following stuff has to be done:

Check if Node installed



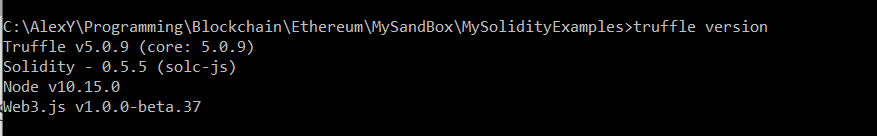
All global packages installed by npm are located here:



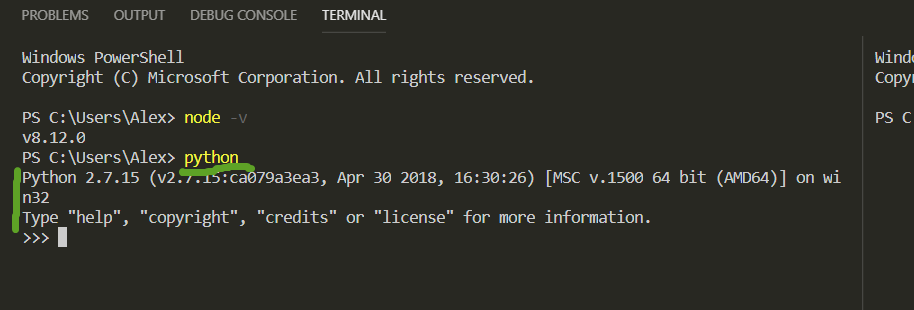
To install Truffle globally run the following command

npm install –g truffle

We can check if the truffle has been install successfully:



Make sure Python 2.7 installed.

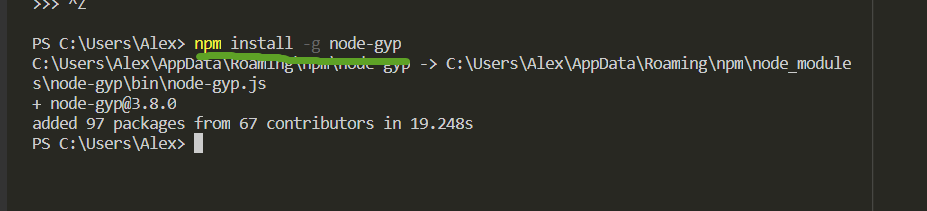


If not, it could be installed from there:

To install it, simply download it from the website: <https://www.python.org/downloads/> .

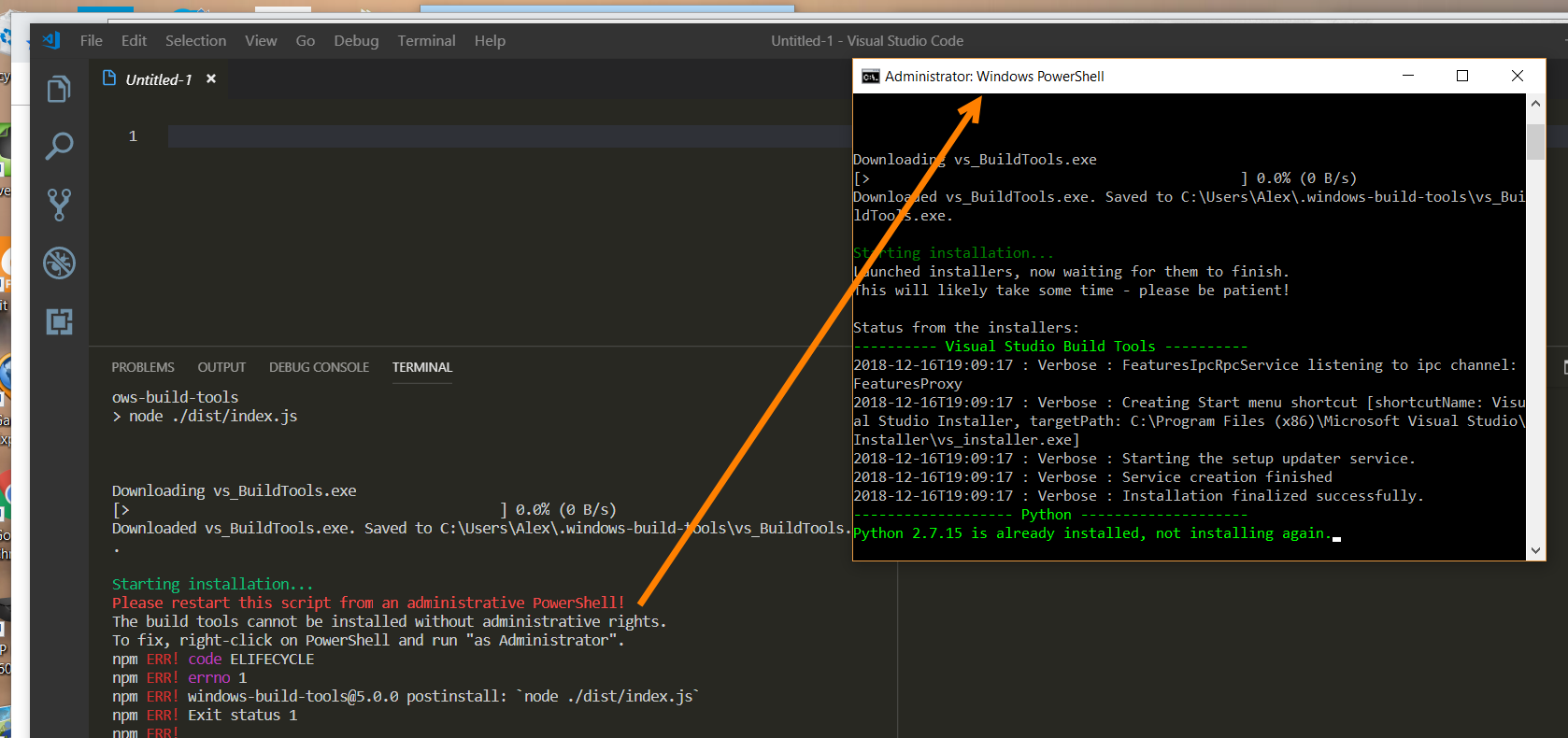
Install node-gyp package.

(Based on the reference link above it is used to compile native Windows addons into node.js



Last install windows-build-tools

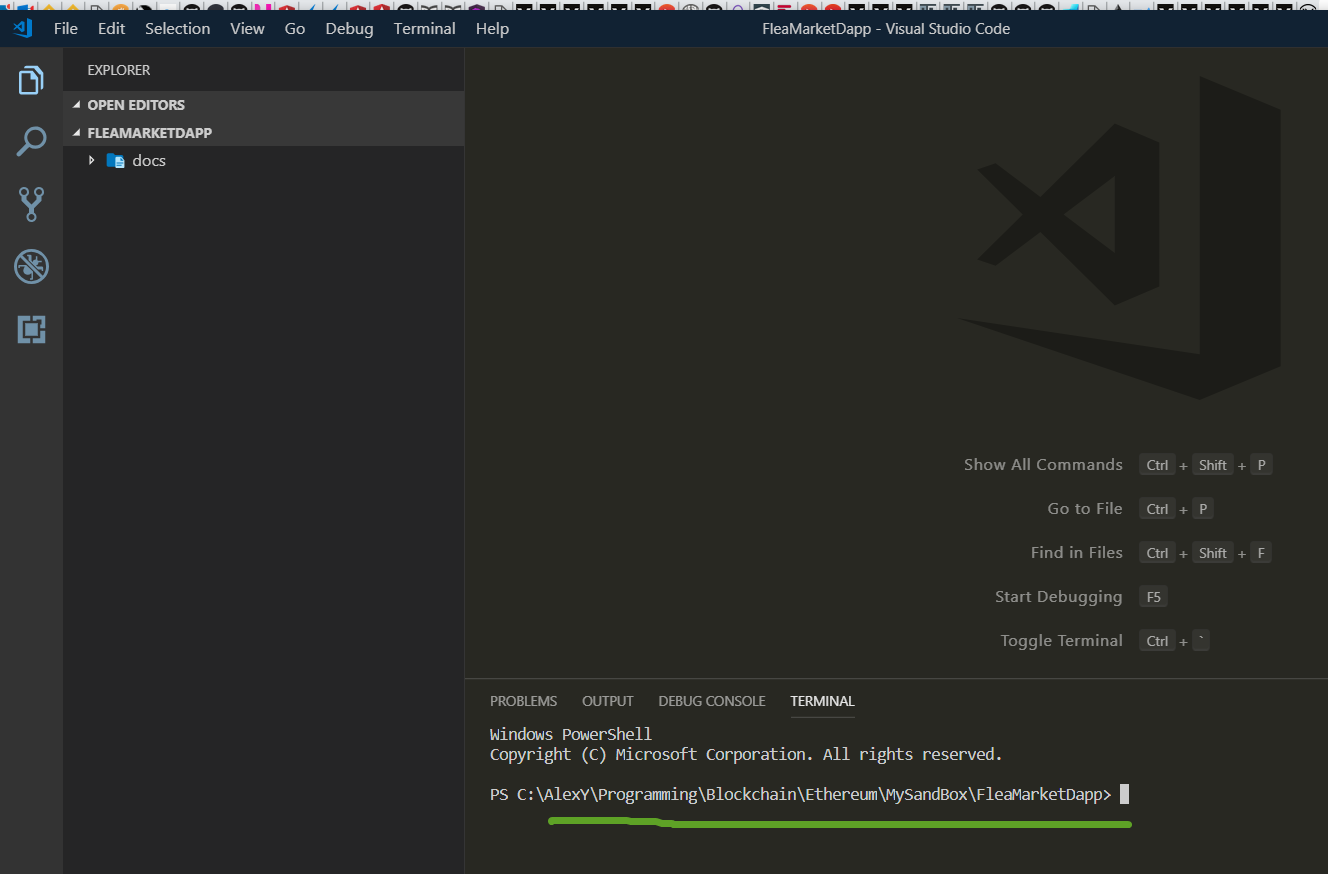
I have to do installation from the cmd windows as Administrator.



Next we create directory

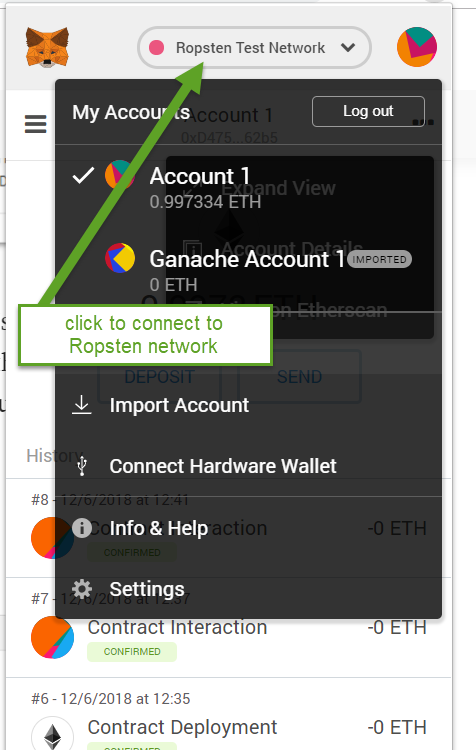


And open VS Code there



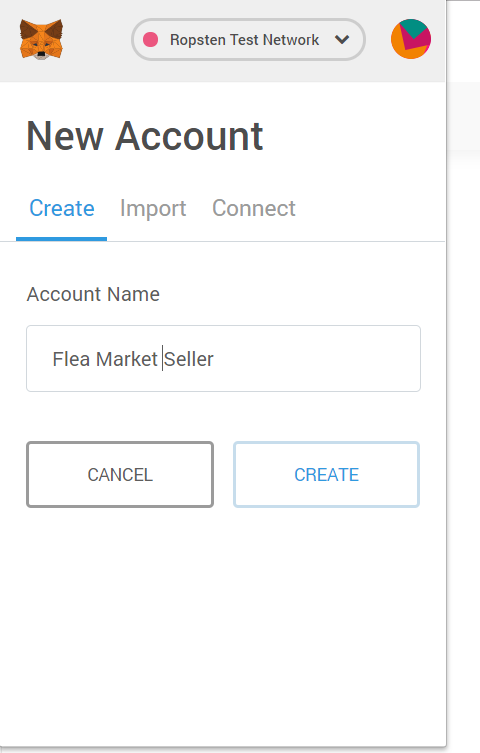
# Guide to deploying smart contracts with Remix and Truffle on Ropsten blockchain

Connect to Ropsten network

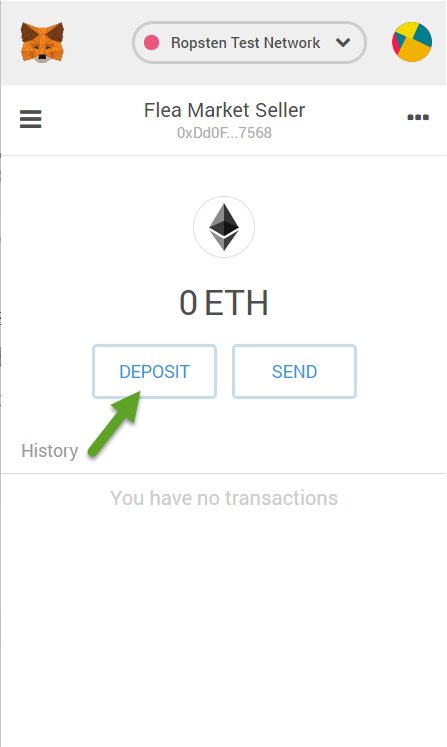


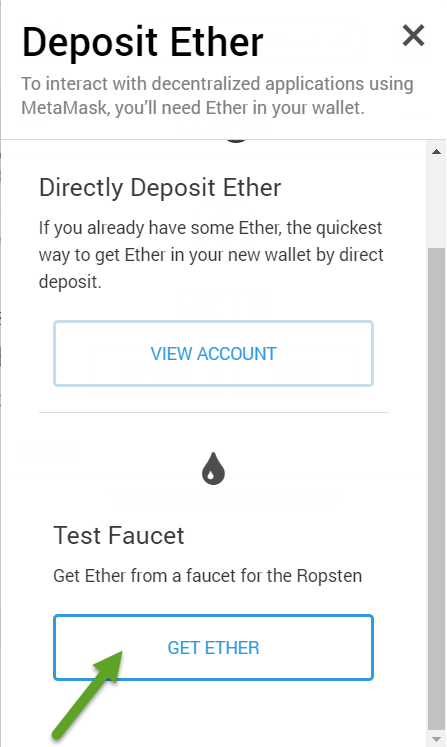
Next let us create a couple account on the Ropsten blockchain network and fill them with some Ethers

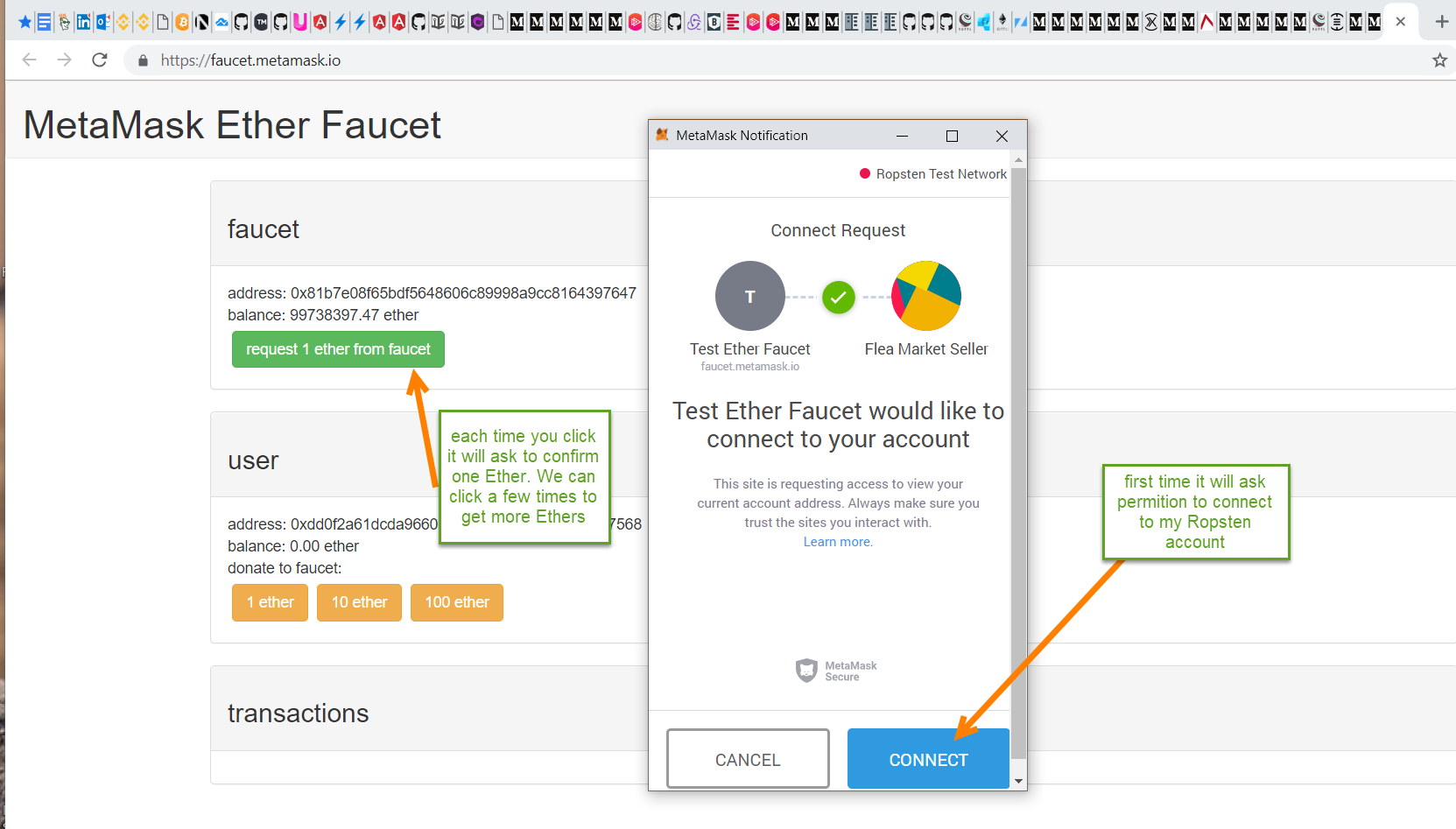
Click add a new account



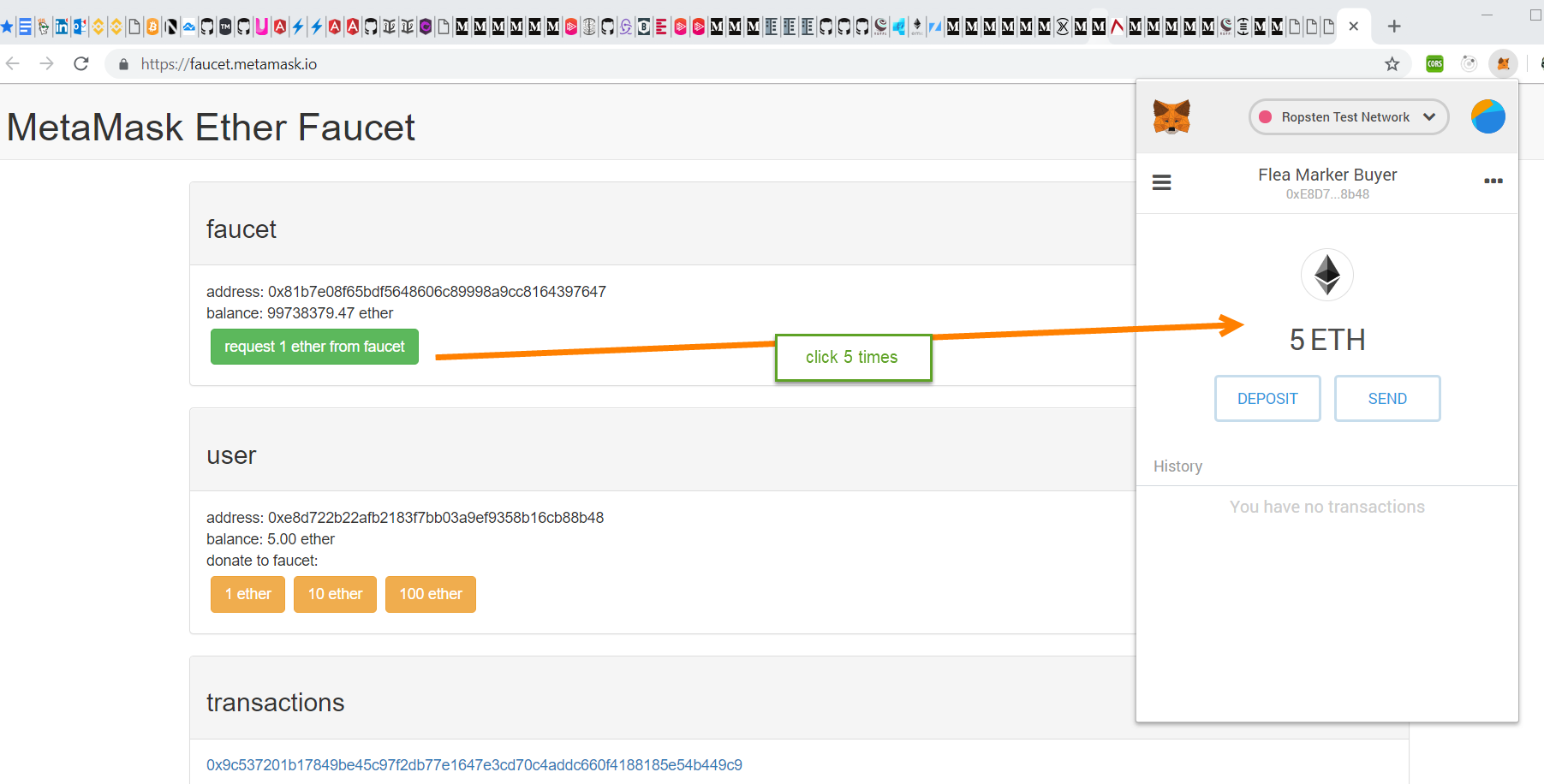
And name it as ‘Flea Market Seller’. Next let add some money to the account



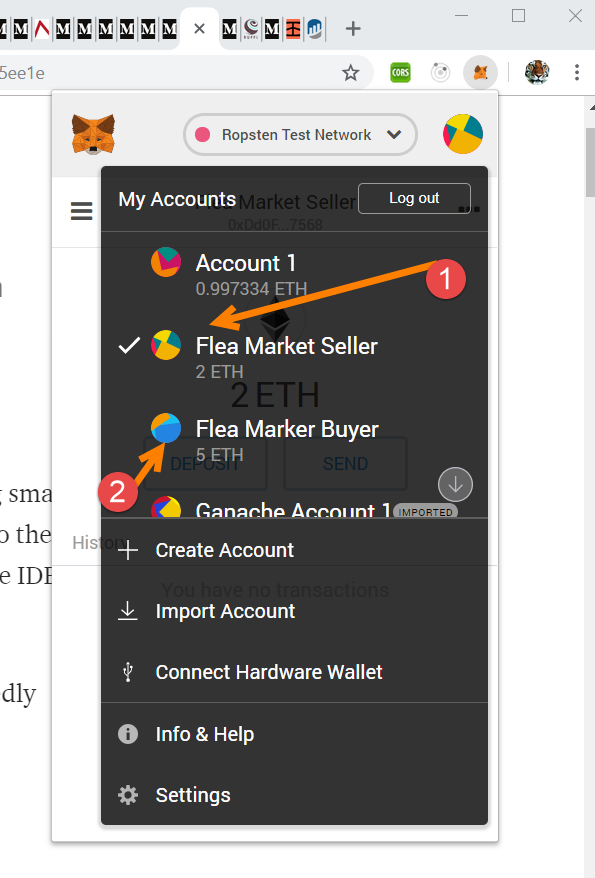




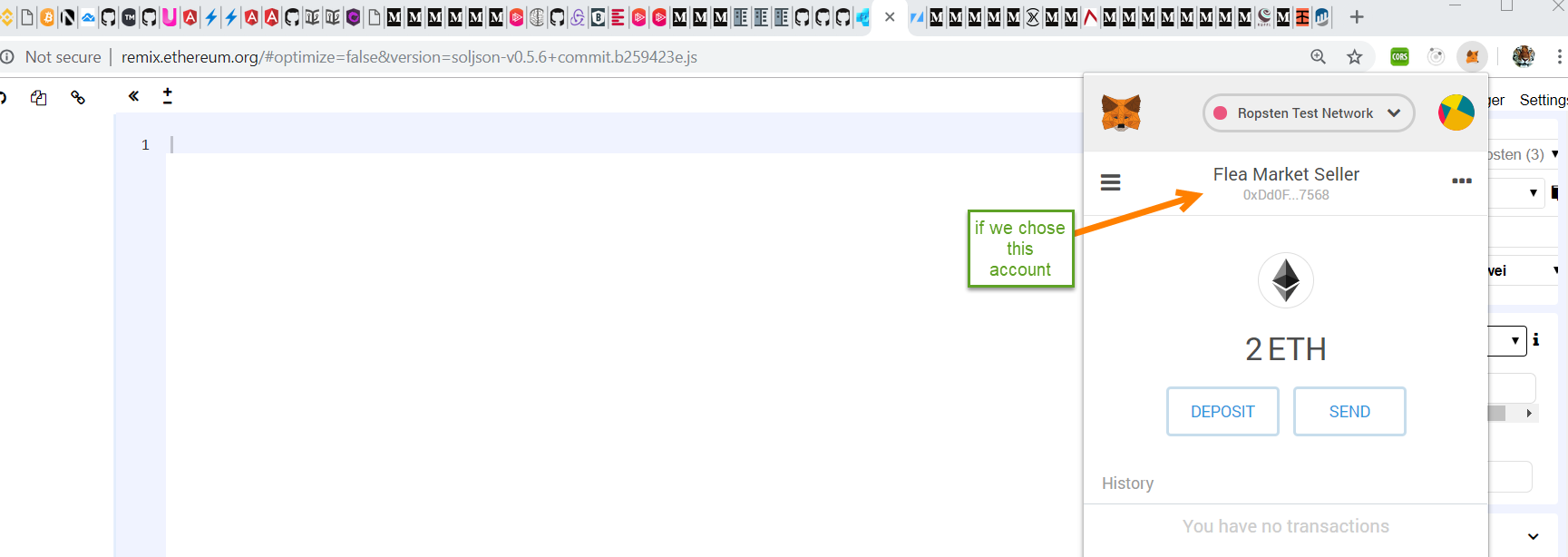
As the result we have



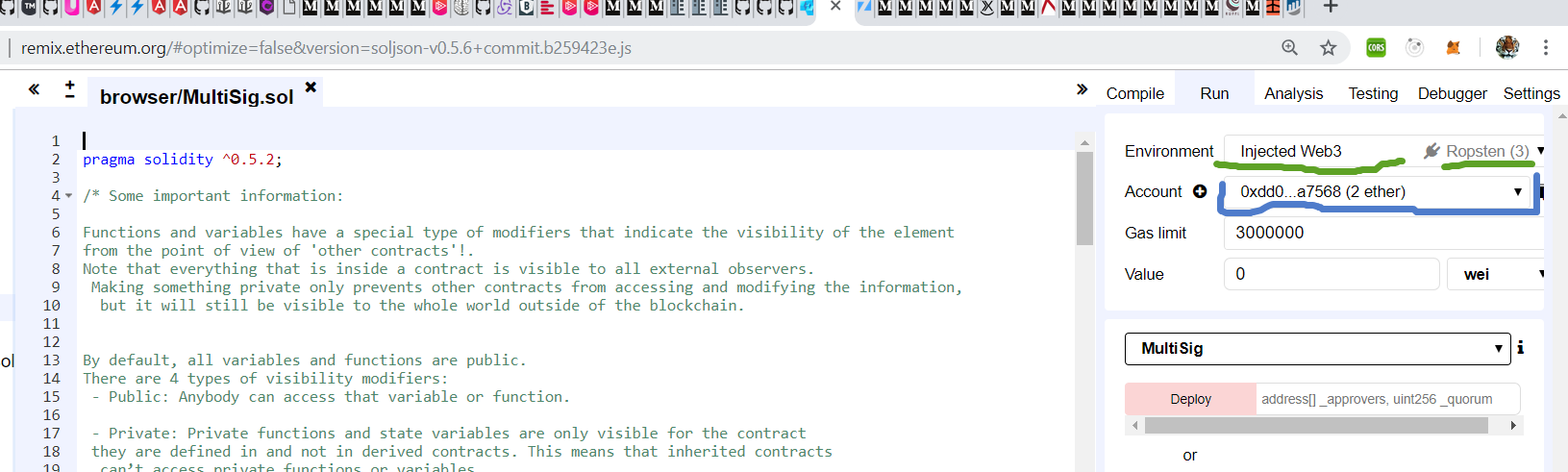
Similar we can create another account:



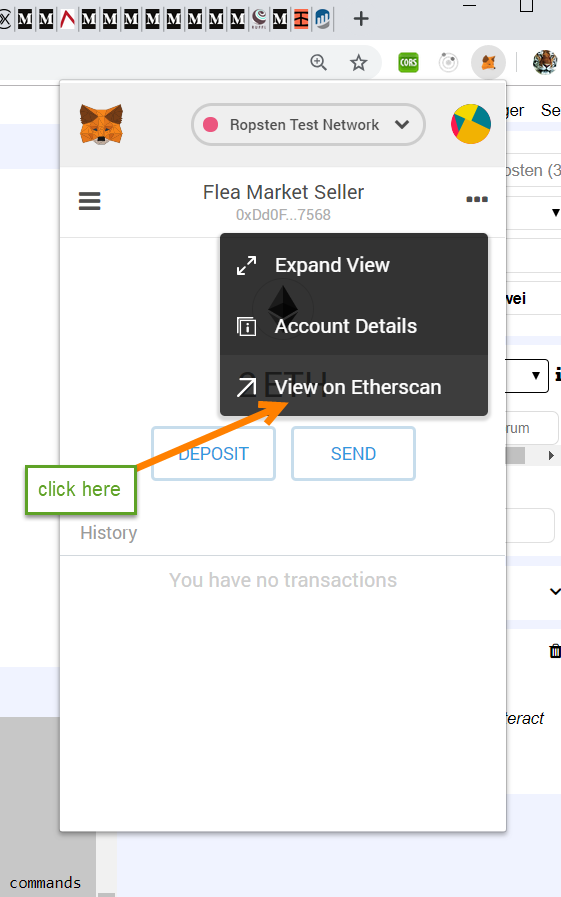
We can use this account with the Remix:

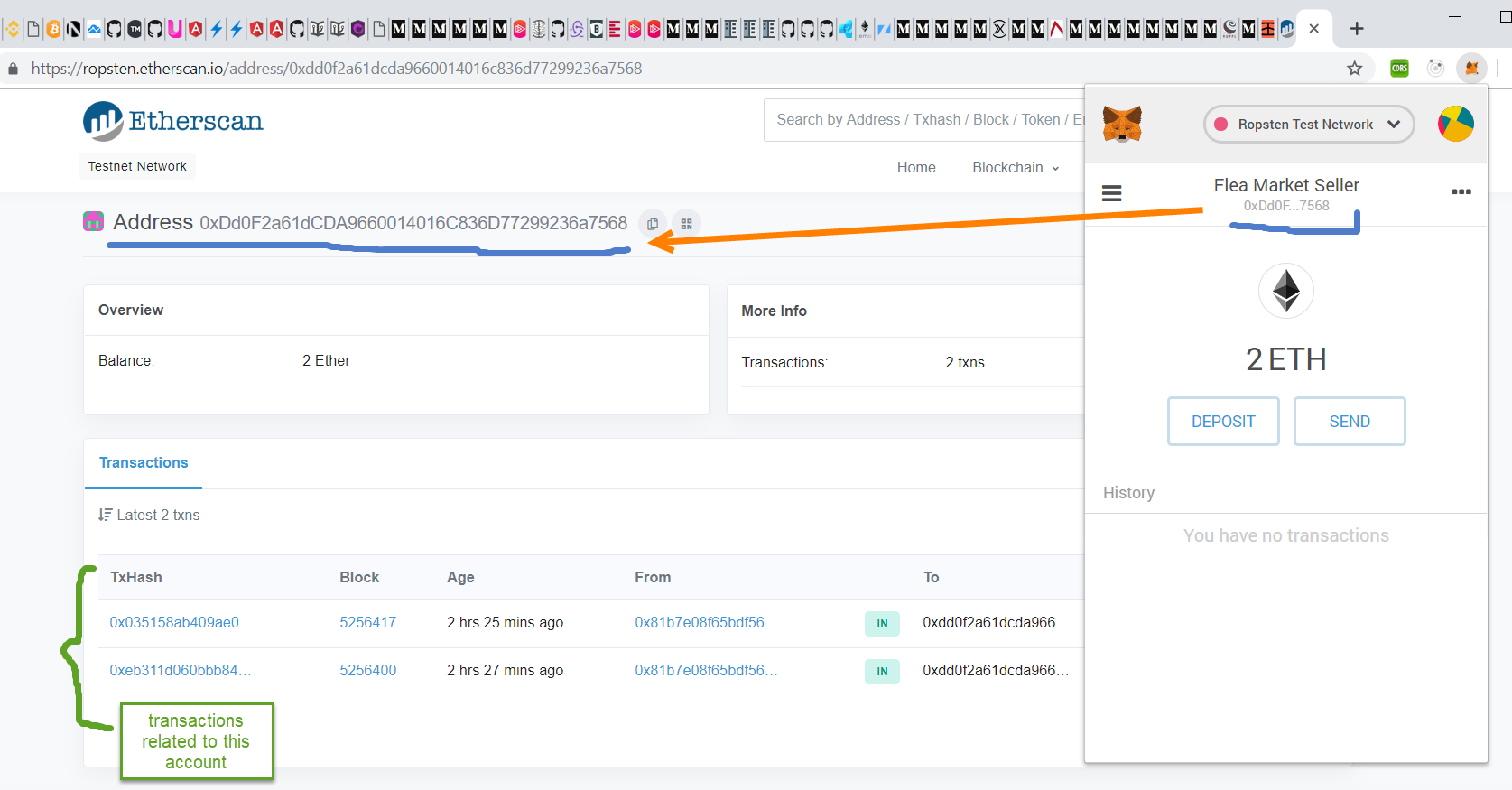


We can see it in the Remix



So we can use Remix to deploy our contract to the Ropsten blockchain. To see all transaction we made on the Ropsten we do the following:





Now let us see how to set up Truffle to be able to deploy on Ropsten network.

Based on

<https://medium.com/coinmonks/5-minute-guide-to-deploying-smart-contracts-with-truffle-and-ropsten-b3e30d5ee1e>

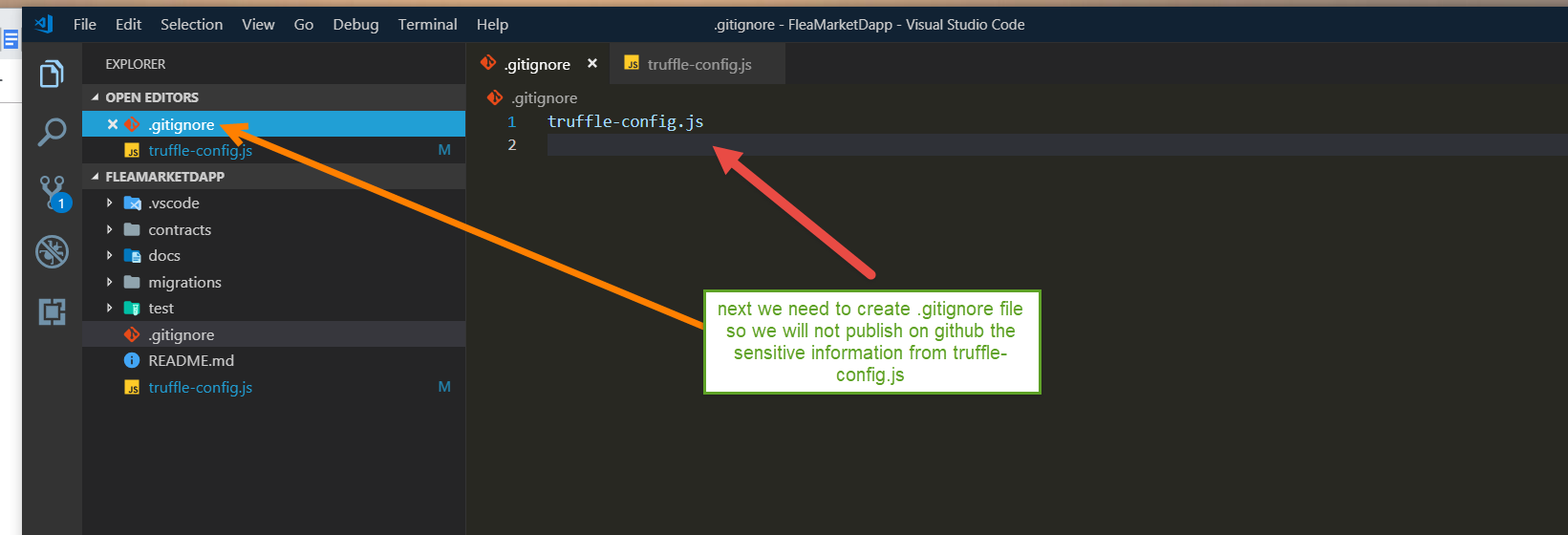
<https://medium.com/coinmonks/steps-to-deploy-a-contract-using-metamask-and-truffle-7ae65e6d8dc8>

<https://truffleframework.com/tutorials/using-infura-custom-provider>

<https://medium.com/coinmonks/rpc-access-to-ethereum-with-infura-318854b7732f>

First we need to create account on Infura:

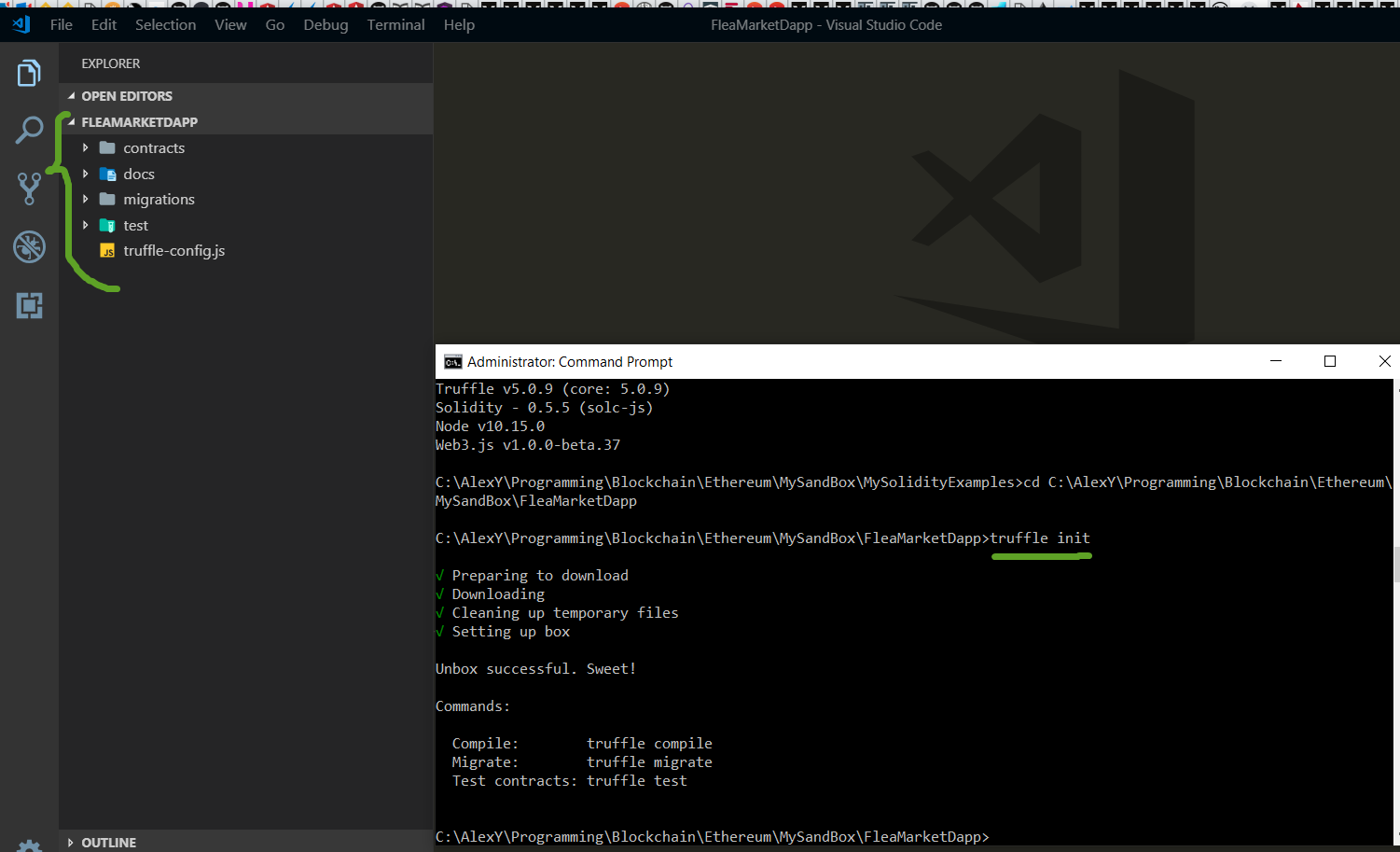
But before let us do this



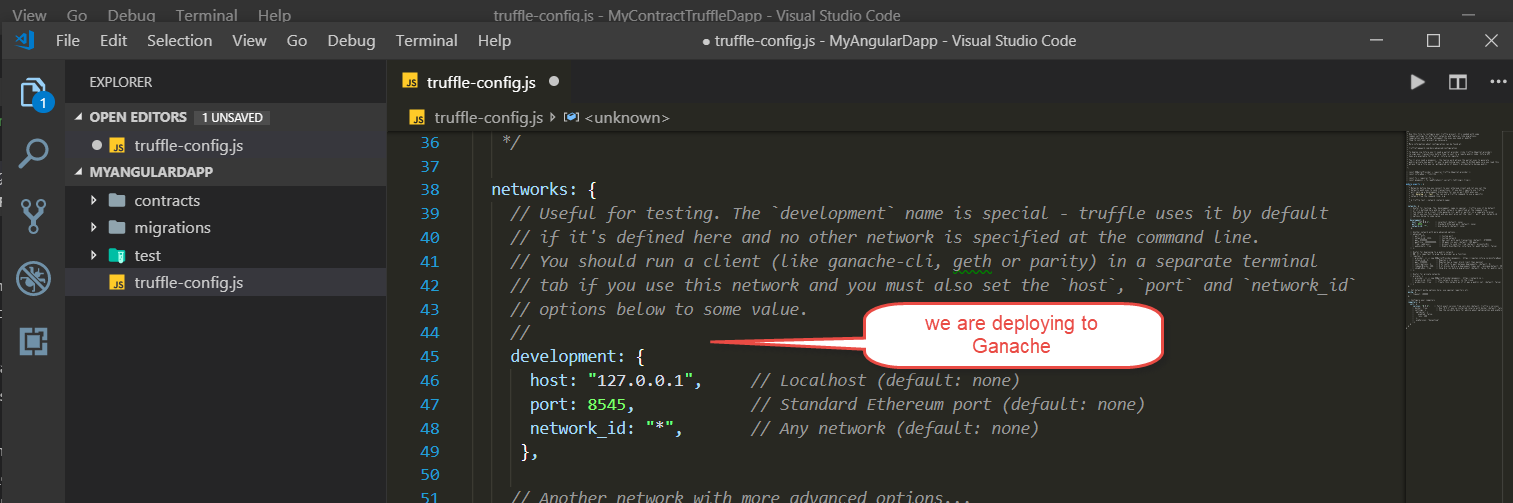
We are going to put the sensitive info into truffle-config.js, so we don’t want to show it on the github.

(!!! Not working, explore this later…)

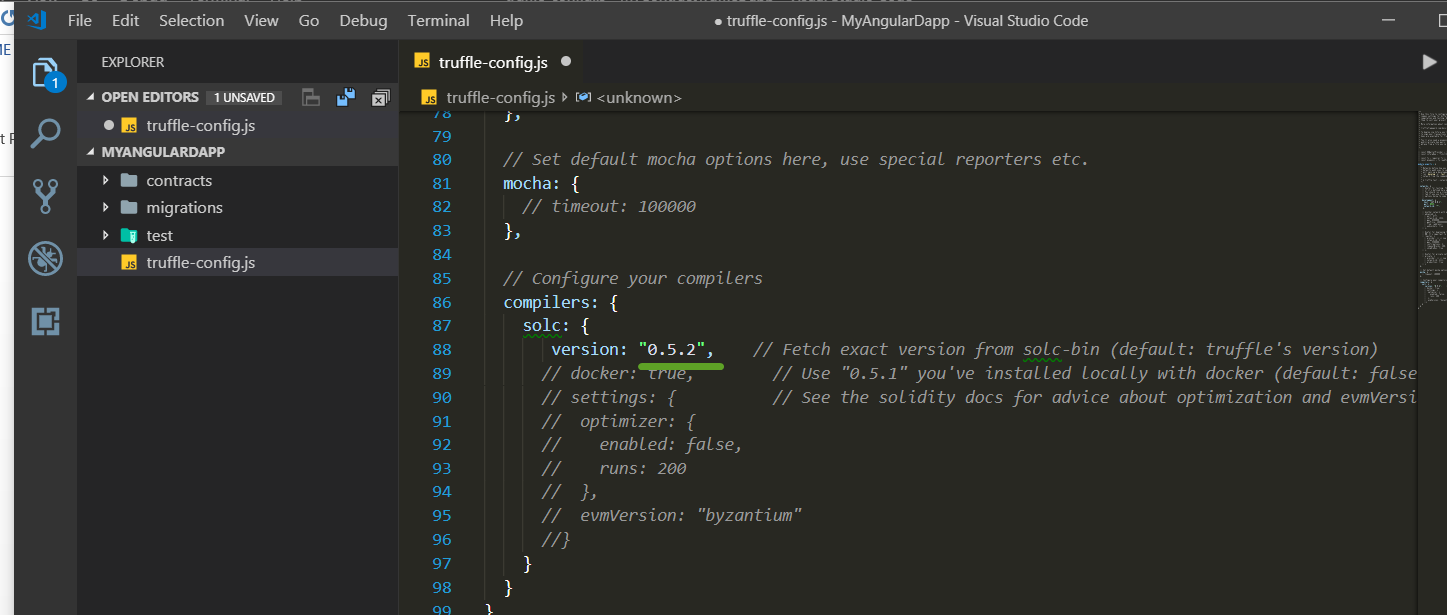
Create truffle project



Next we modify truffle-config.js to tell where we are going to deploy contract



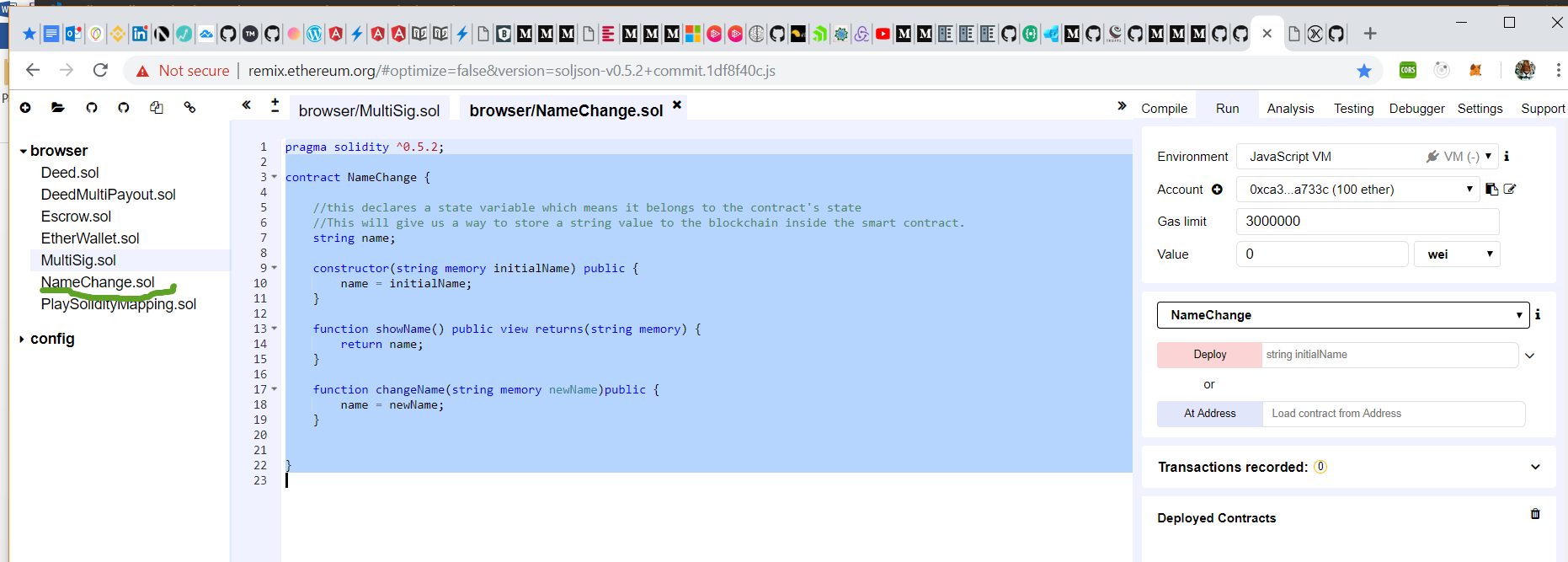
And we also specify what compile version of Solidity to use to compile the future contract



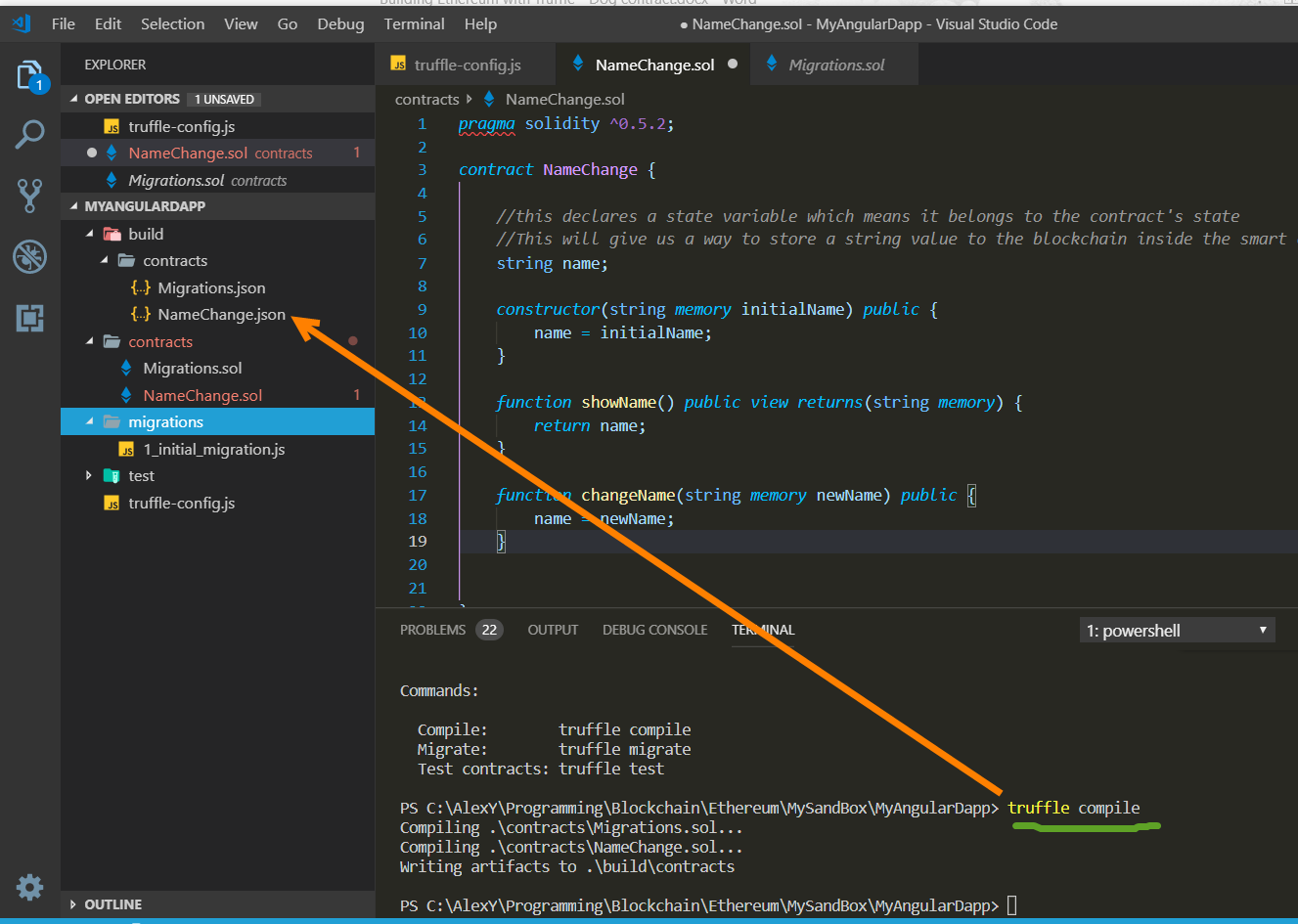
Next we create a simple contract



It is always good idea first compile and run contract in Remix, so we sure all good with Solidity



Now we are ready to compile our contract



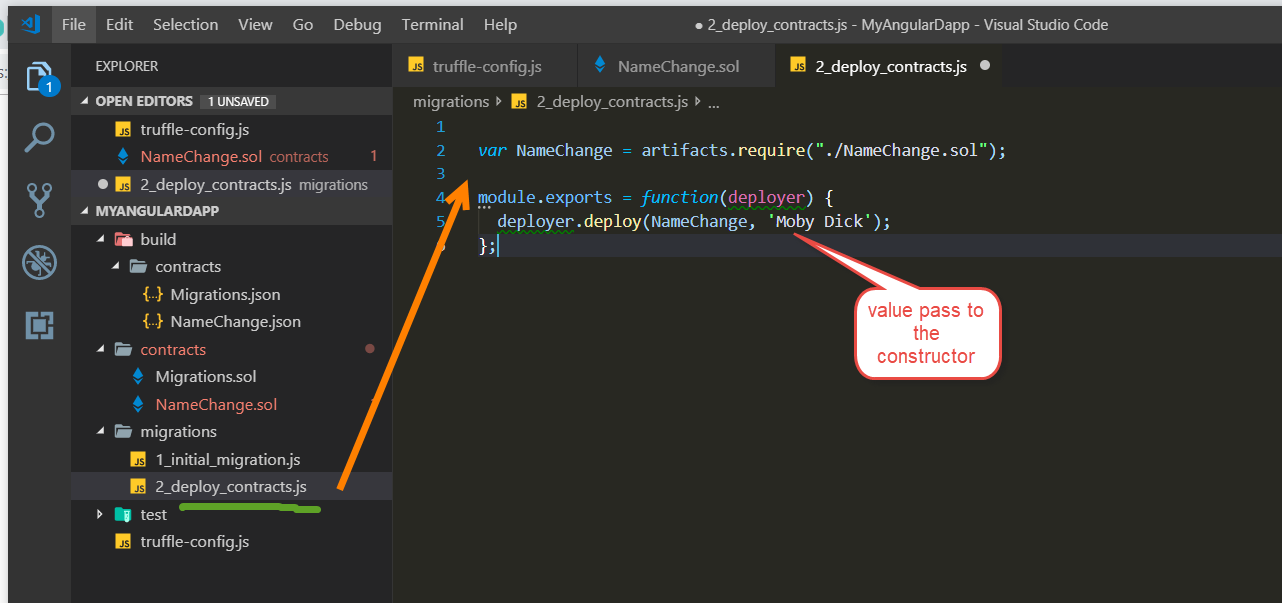
Which generate the corresponding NameChange.json file:

This file is the smart contract ABI file, which stands for "Abstract Binary Interface". This file has many responsibilities, but two that I will highlight here:

* It contains the compiled bytecode version of the Solidity smart contract code that can be run on a Ethereum Virtual Machine (EVM), i.e., an Ethereum Node.
* It contains a JSON representation of the smart contract functions that can be exposed to external clients, like client-side JavaScript applications.

Now we are ready to create our own migration script.

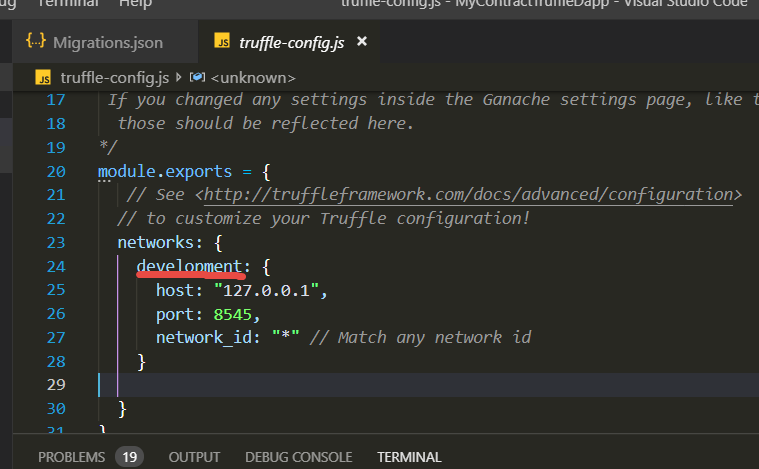
1. Create a new file named **2\_deploy\_contracts.js** in the **migrations/** directory.
2. Add the following content to the **2\_deploy\_contracts.js** file:

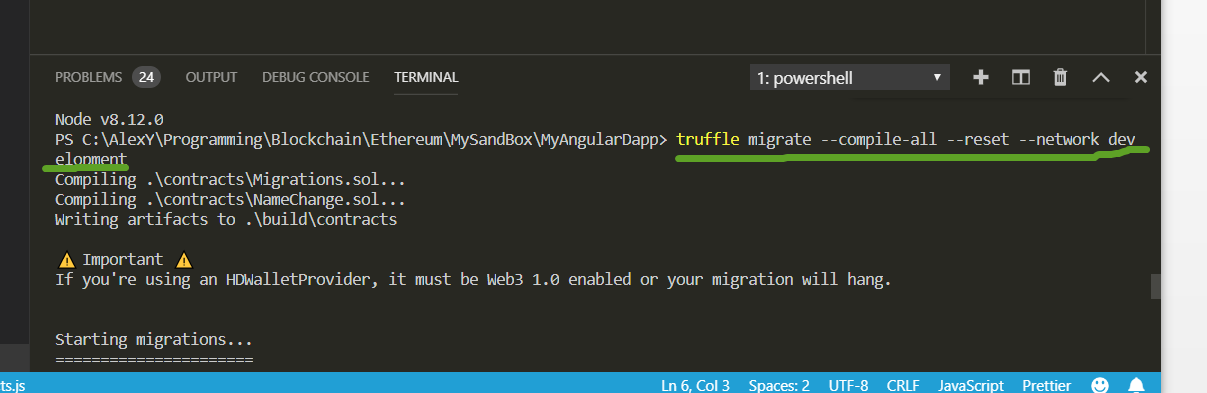


Next we deploy our contract to Ganache blockchain

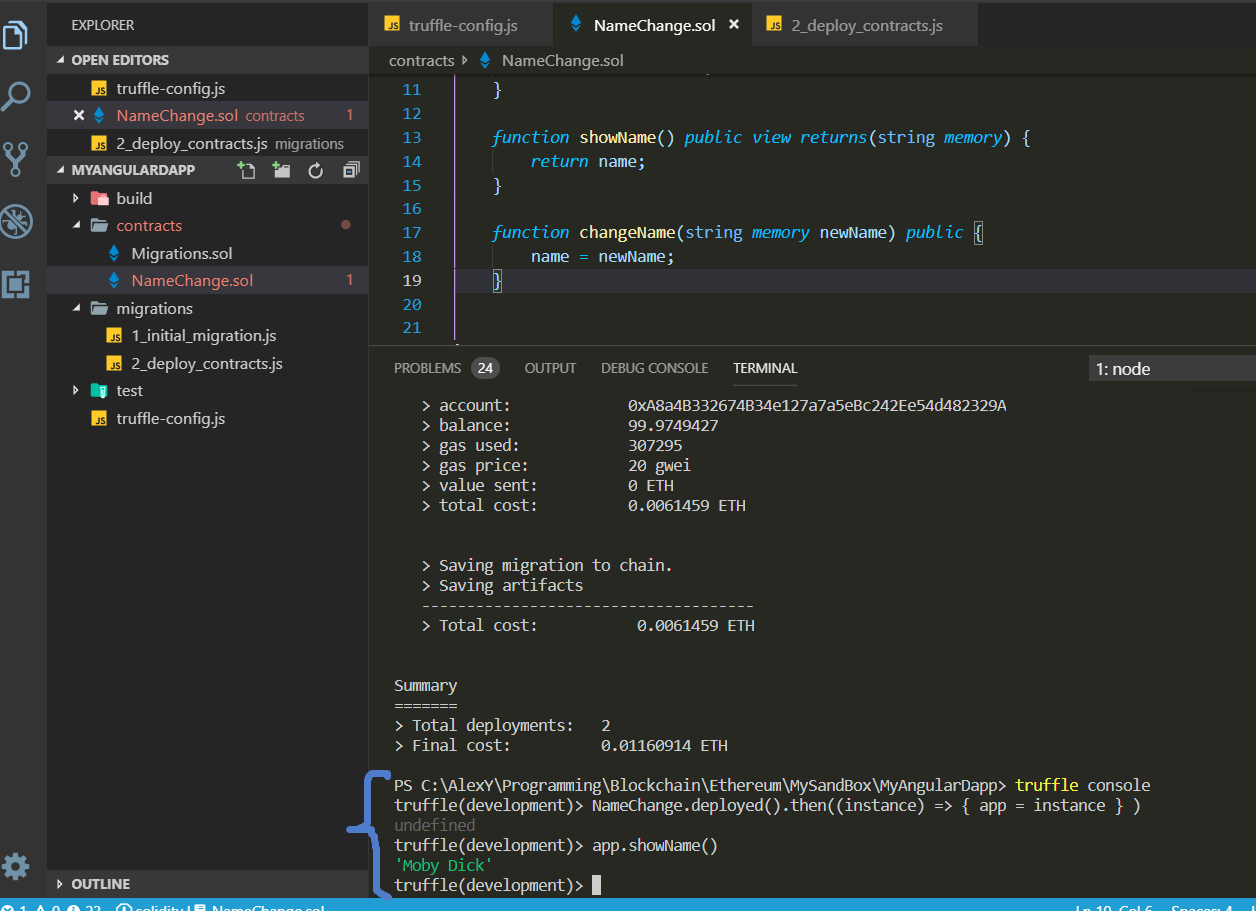
* truffle migrate --compile-all --reset --network ganache

Here the name ‘development’ is the name of the network running our blockchain:





Let us check that the contact was deployed successfully. Switch to the truffle console

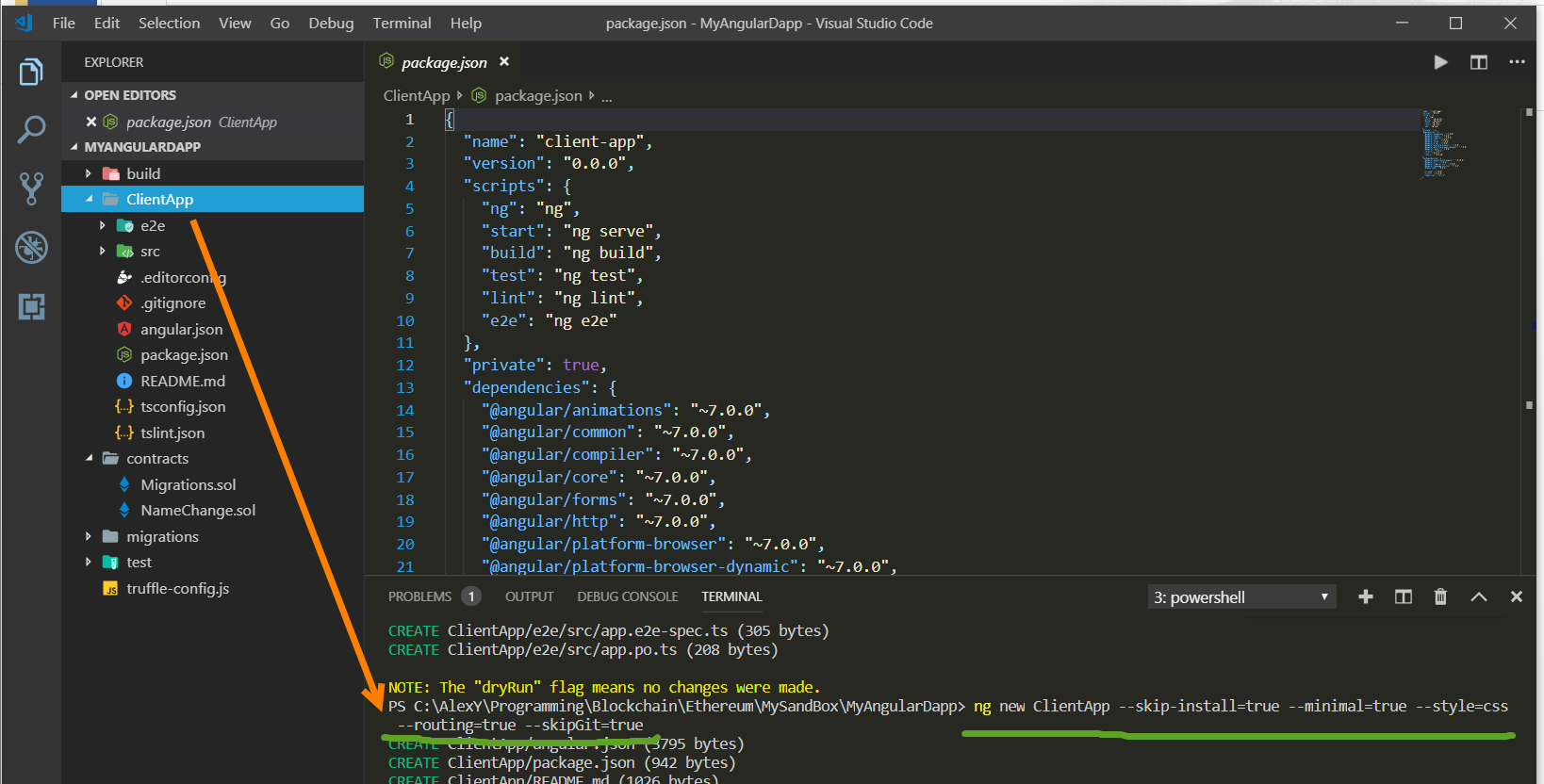


We can exit the Truffle console by typing this command:

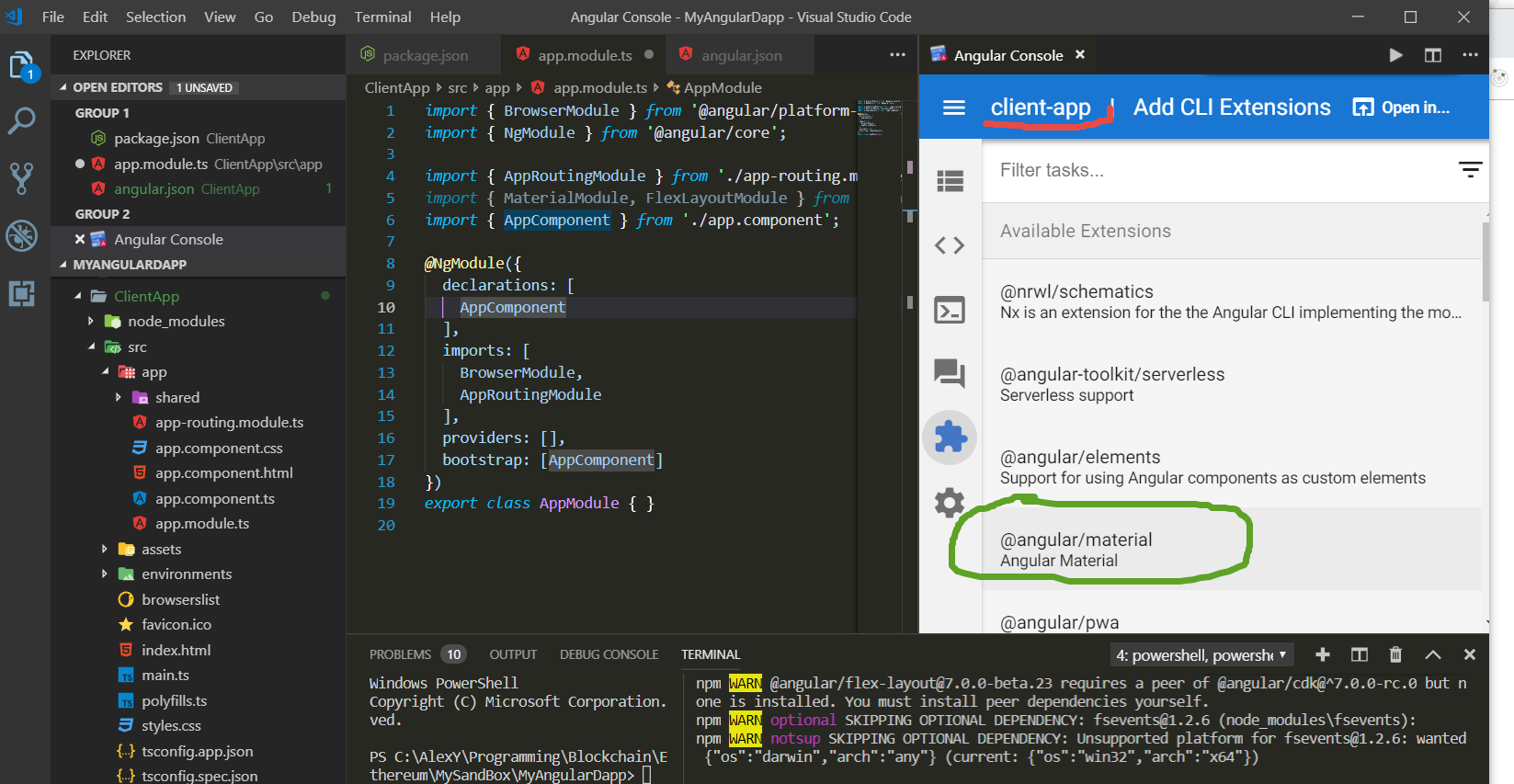
.exit

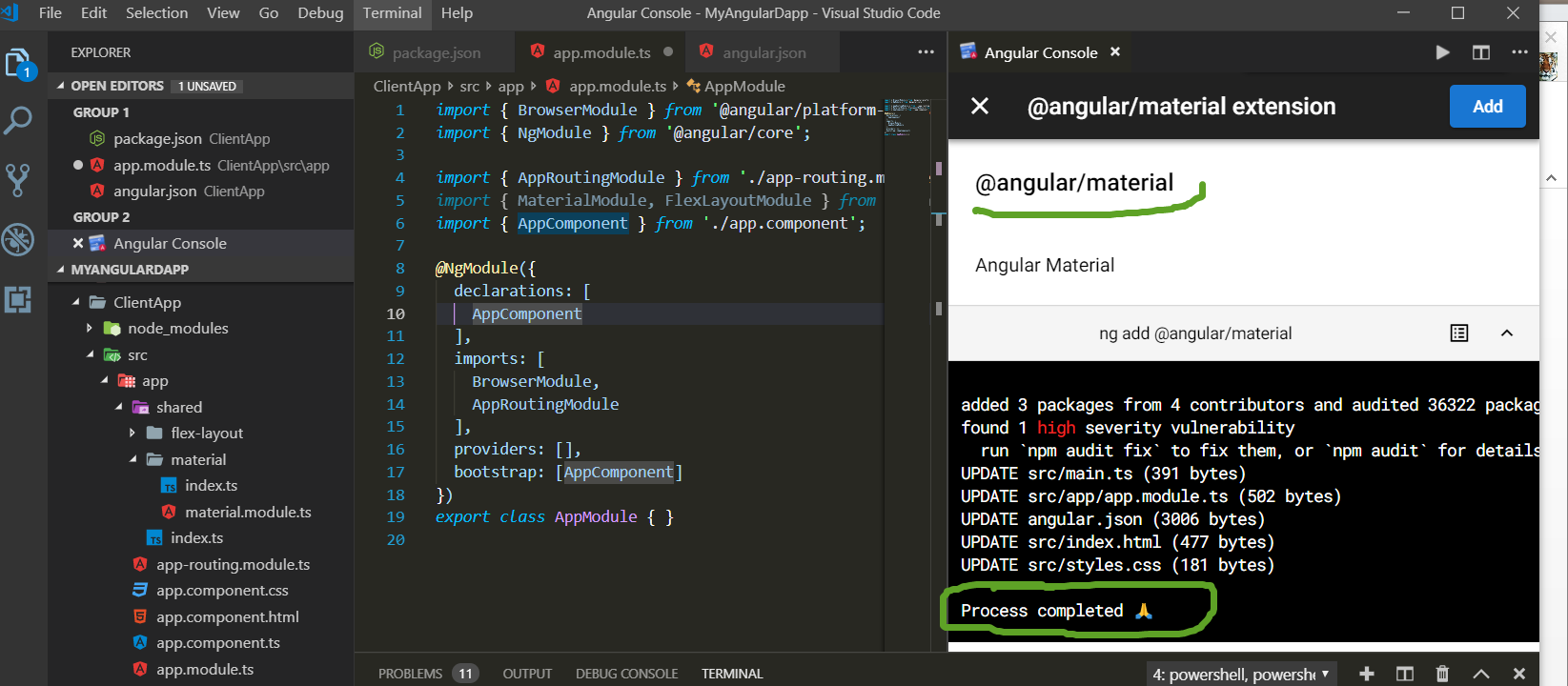
Create Angular project

Best to create the angular app in the separate folder:

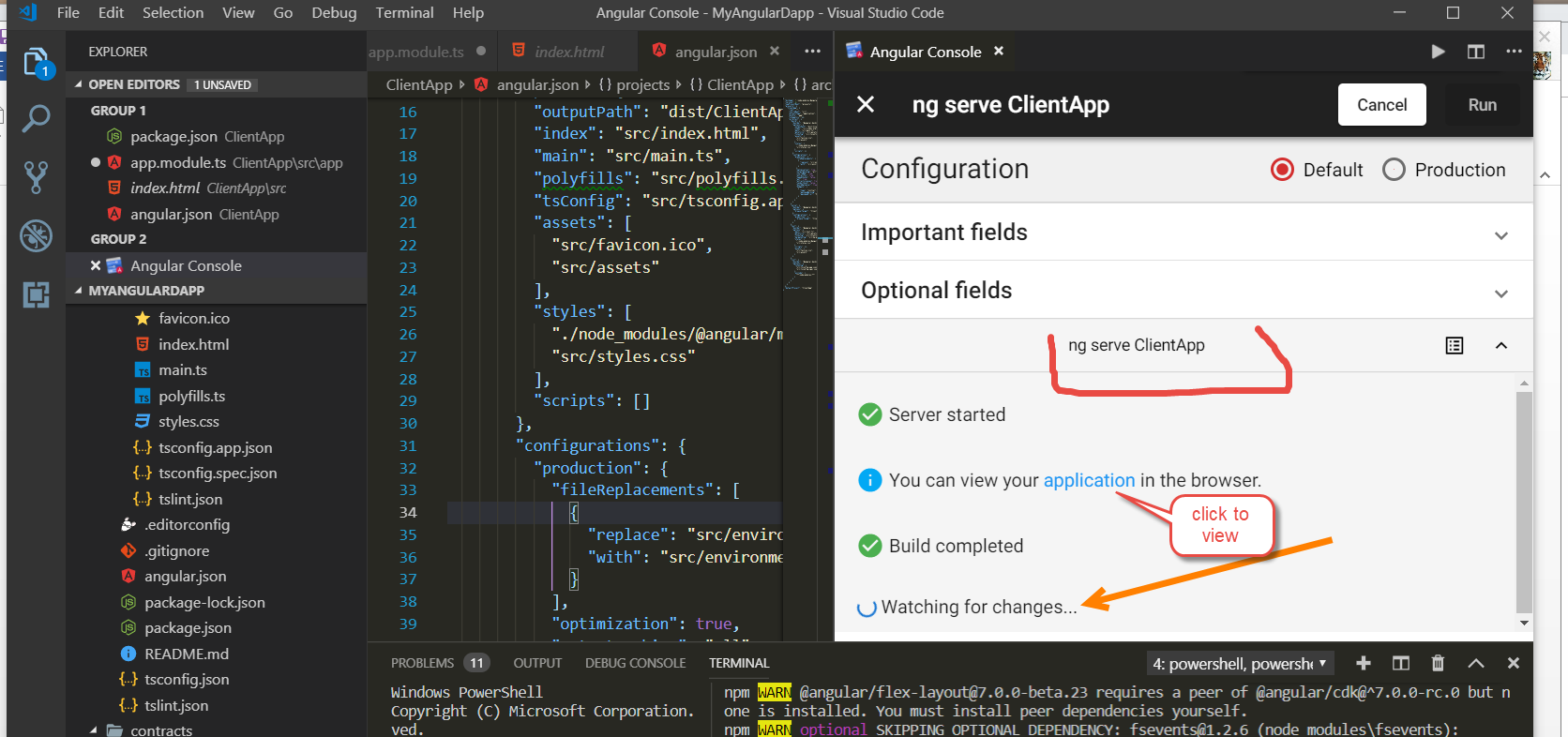


Next using Angular Console let us add angular material package.





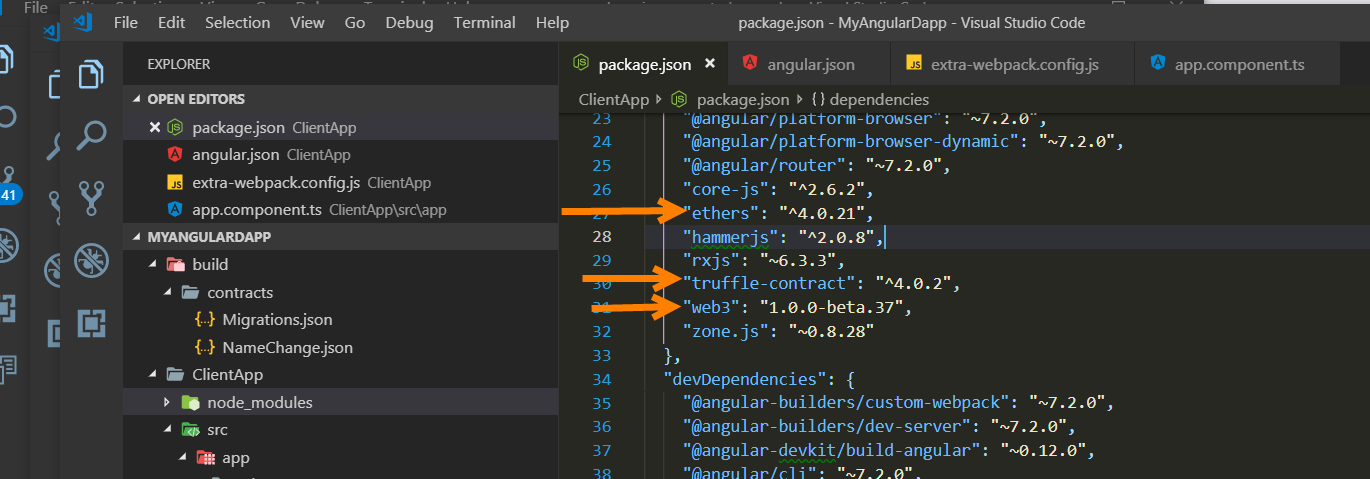
Now let us run Angular app



So we see the result:



Next we install our smart contract packages:

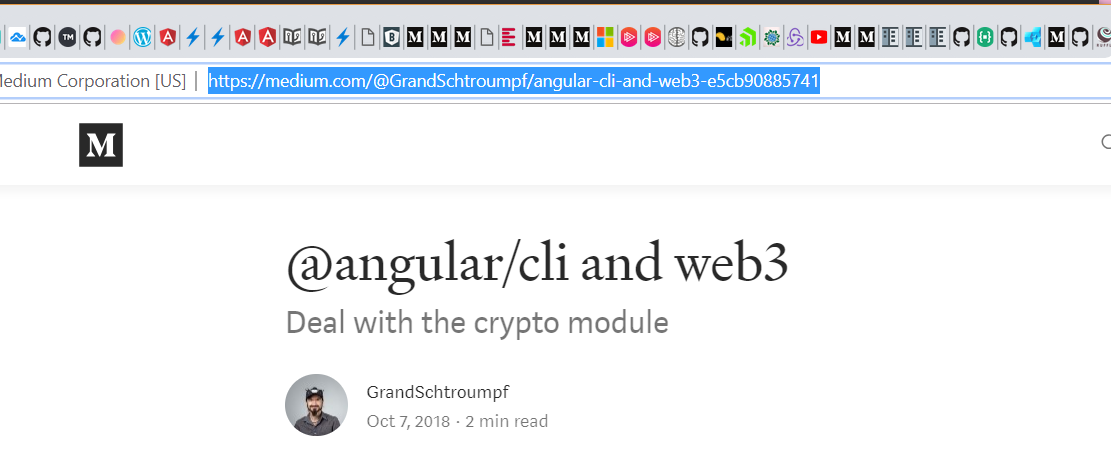


If we try to compile our project we will get some error like this:

Module not found: Error: Can’t resolve ‘crypto’

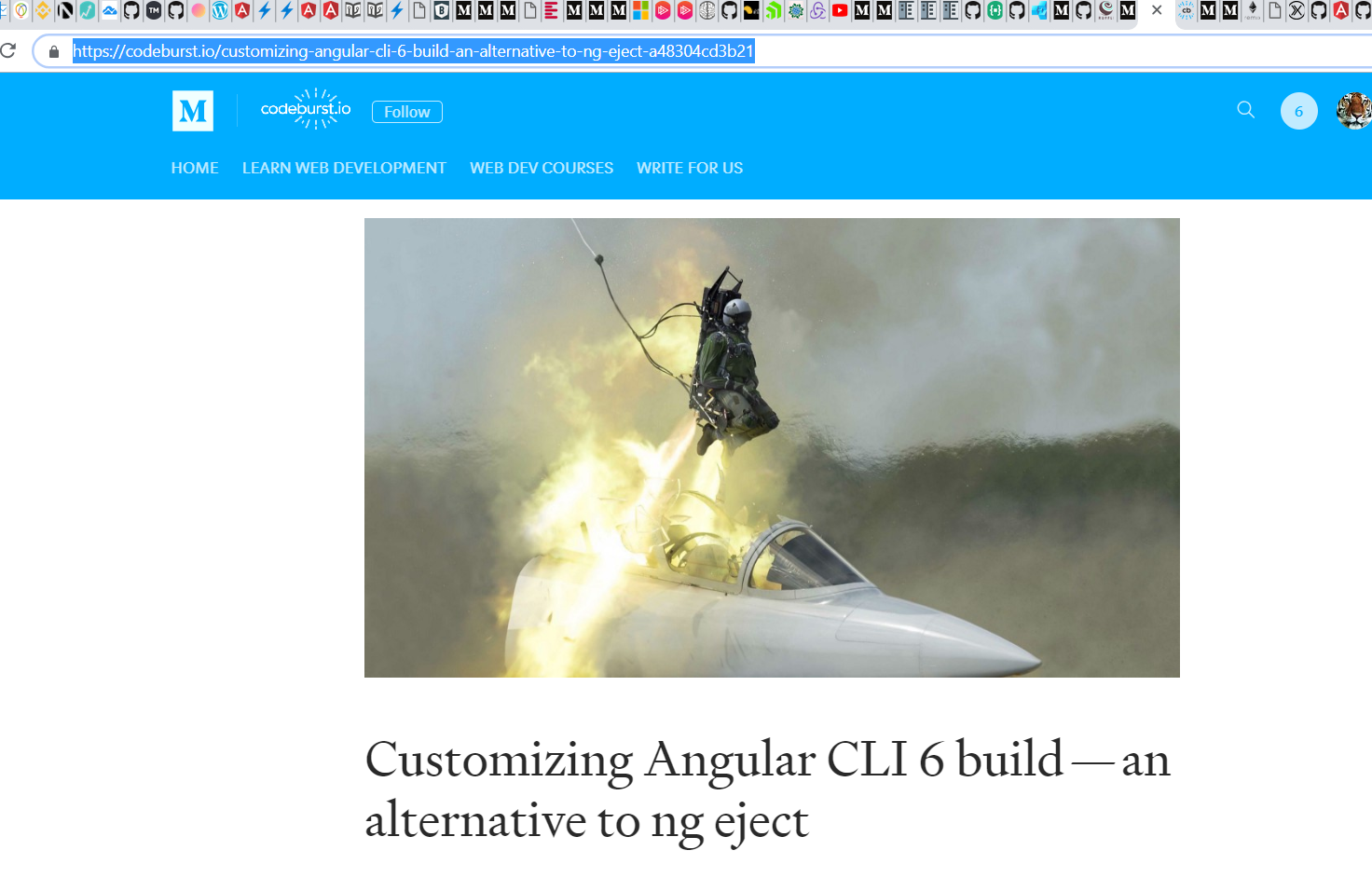
To overcome this problem we have to follow

<https://medium.com/@GrandSchtroumpf/angular-cli-and-web3-e5cb90885741>

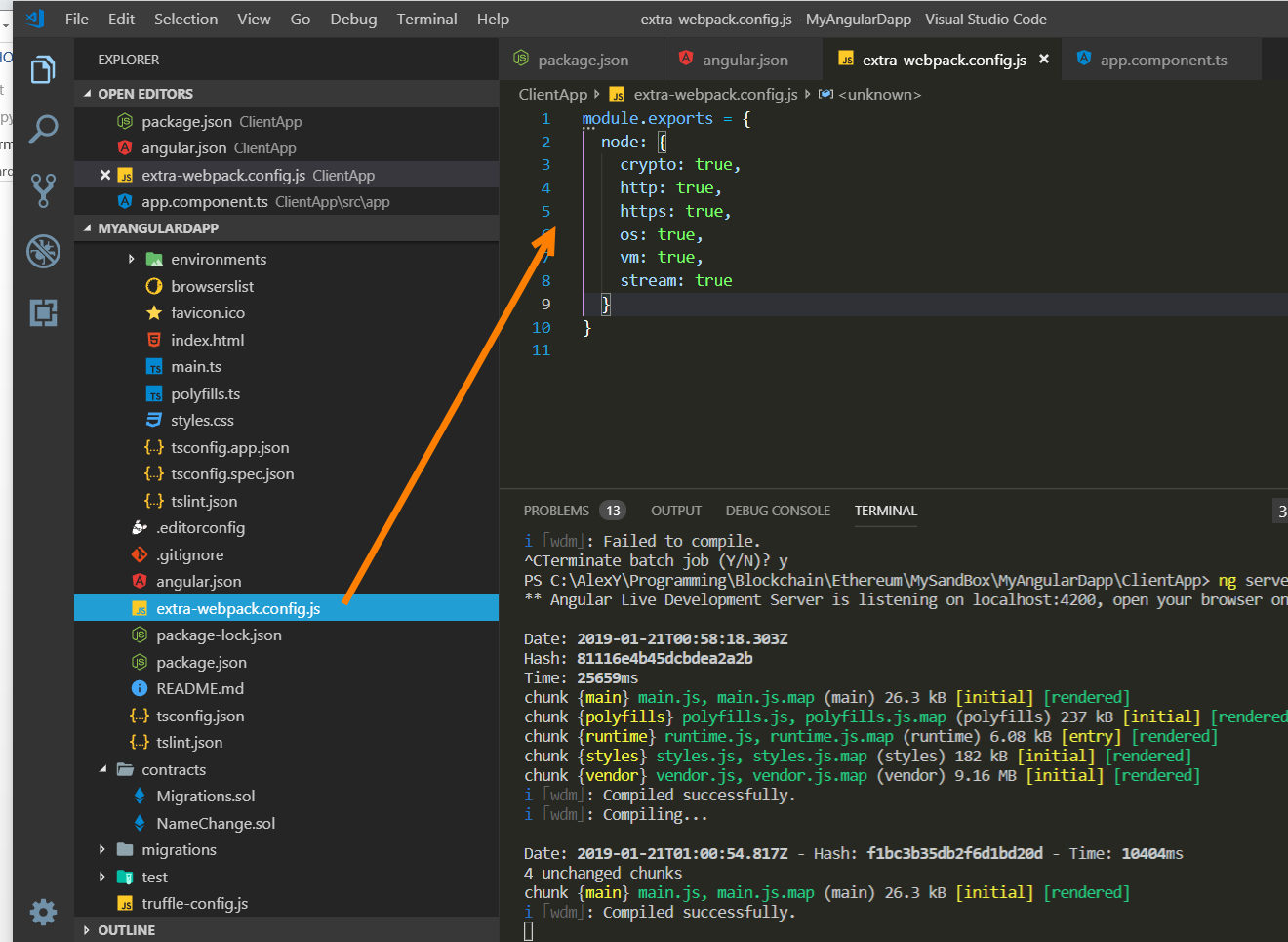


But even better is to use what is described here:

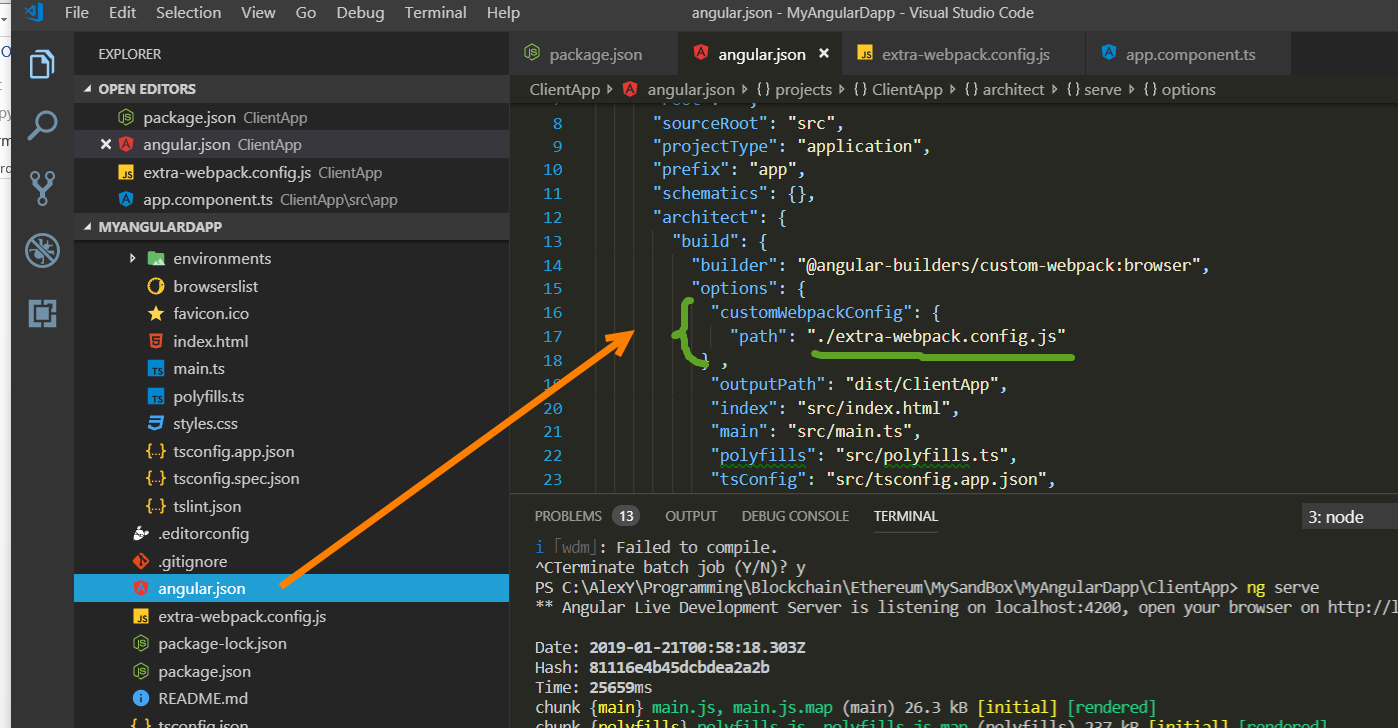
<https://codeburst.io/customizing-angular-cli-6-build-an-alternative-to-ng-eject-a48304cd3b21>



Basically we have to install recommended packages and add the file:



This custom ‘merger’ to webpack.config.js is referred in angular.json file:



After that we don’t have any problem to compile.

Approach I. Simple setup

This one is based on <https://medium.com/b2expand/inject-web3-in-angular-6-0-a03ca345892>

We just use the following code

*// !!Example of using web3 without injecting server or NgRx*

*import* { Component, OnInit, Inject } *from* '@angular/core';

*// Web3*

*import* { WEB3 } *from* './ethereum/tokens';

*import* Web3 *from* 'web3';

*@*Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

*export* *class* AppComponent *implements* OnInit {

title = 'Super Angular with Smart Contract';

*constructor*(*@*Inject(WEB3) *private* web3: Web3) {}

*/\**

*based on https://medium.com/metamask/https-medium-com-metamask-breaking-change-injecting-web3-7722797916a8*

*and*

*based on https://medium.com/b2expand/inject-web3-in-angular-6-0-a03ca345892*

*This code use the new way to connect to the MetaMask.*

*!!!The first time you use this token in your code you should call the enable method of window.ethereum :*

*\*/*

*async* ngOnInit() {

*if* ('enable' in *this*.web3.currentProvider) {

*try* {

*// Request account access if needed*

*await* *this*.web3.currentProvider.enable();

*// Accounts now exposed*

*/\* do test call*

*this.web3.eth.sendTransaction({*

*//*

*});*

*\*/*

} *catch* (error) {

console.log('can not access web3 account', error);

}

}

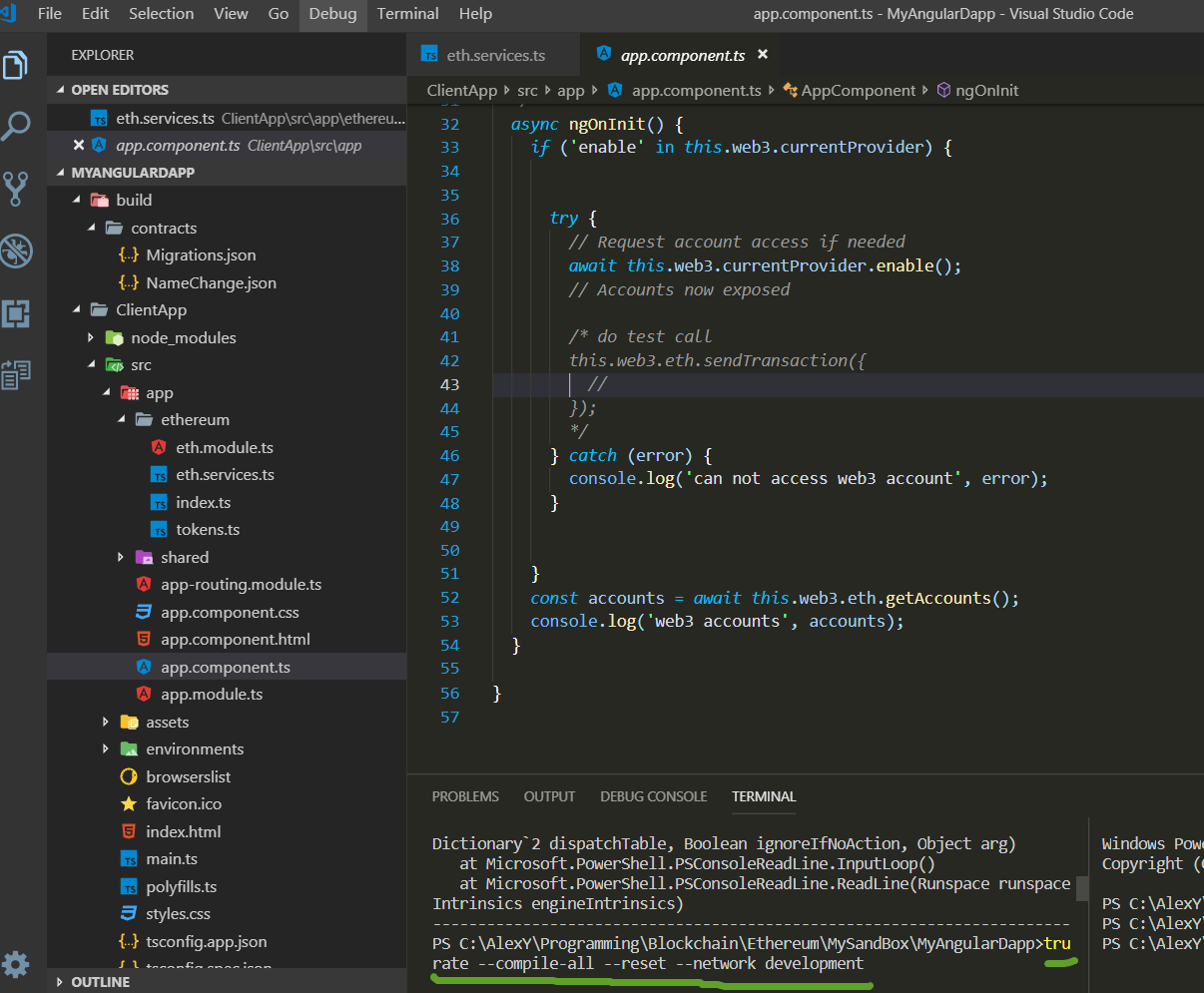
*const* accounts = *await* *this*.web3.eth.getAccounts();

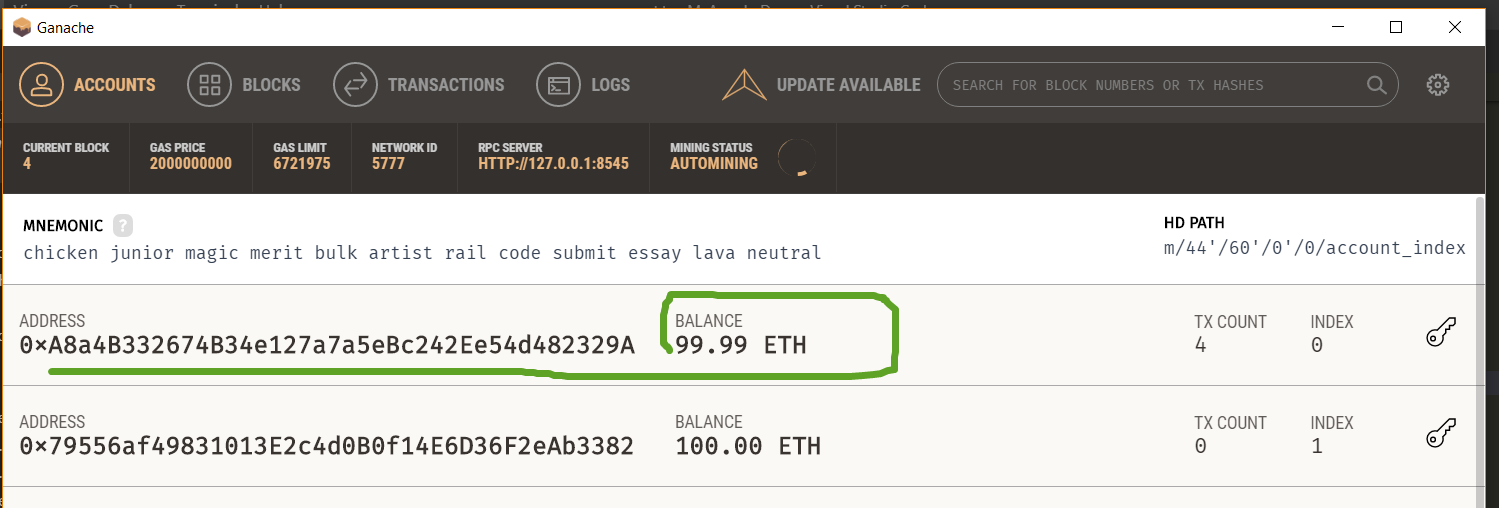
console.log('web3 accounts', accounts);

}

}

Let us deploy contract on the Ganache





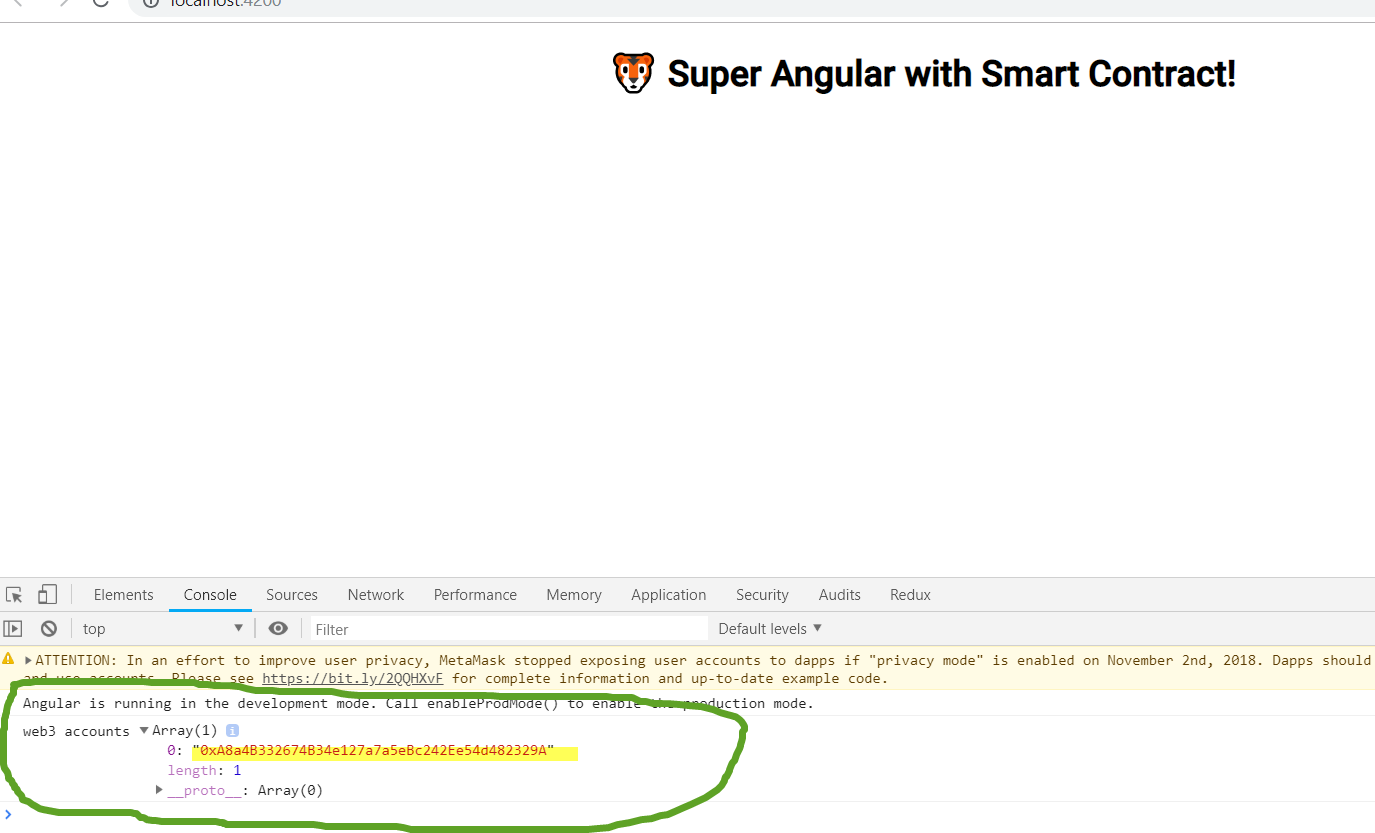
And connect to MetaMask



After we compile and run the Angular app we have



We click Connect and we see the output displaying the account:



Approach II Using NGRX

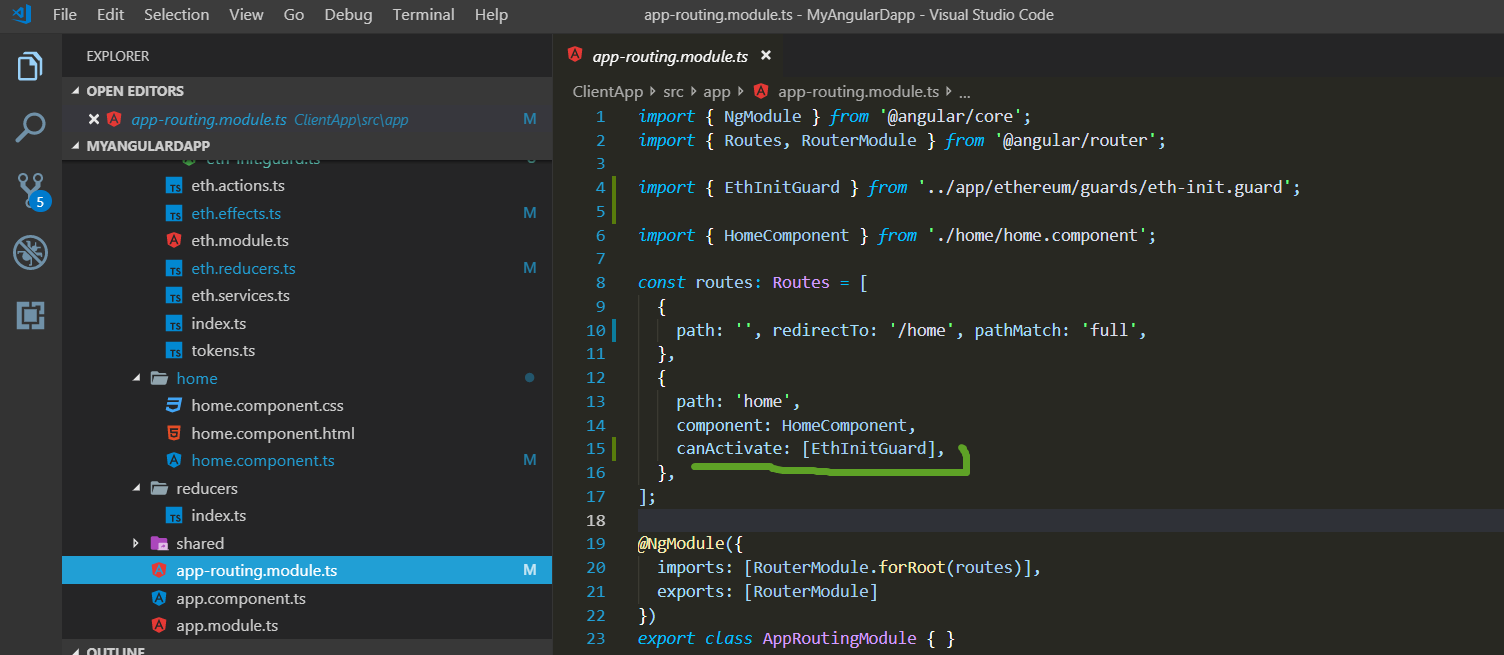
We follow the main document site https://ngrx.io/guide/store

Let install the following libraries

npm install @ngrx/store, @ngrx/effects, @ngrx/entity, @ngrx/store-devtools} –save

npm install @ngrx/router-store, ngrx-store-freeze --save

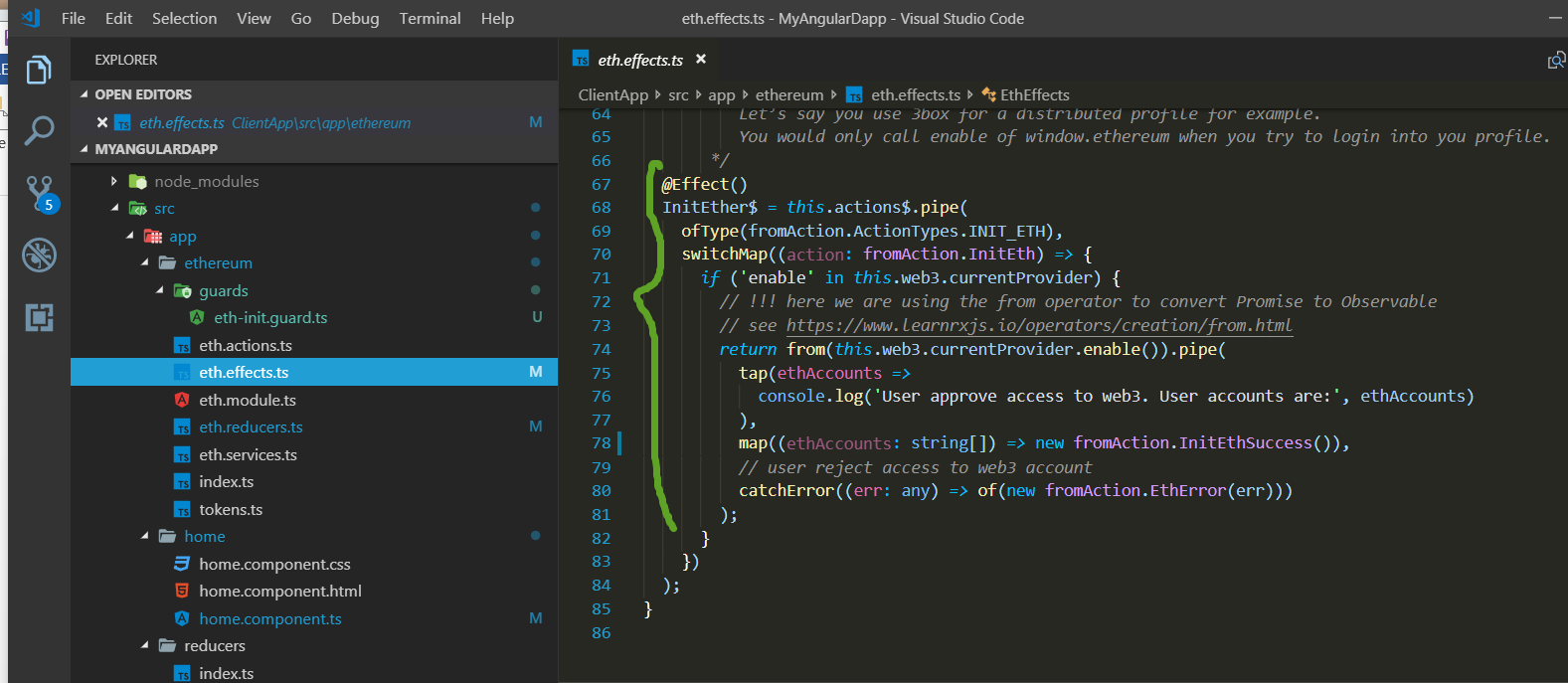
The most interesting moment here how we implemented the ether initialization. We use the guard -> effect to make sure the ether get initialized before we even move to the home page:



The guard:



The effect:

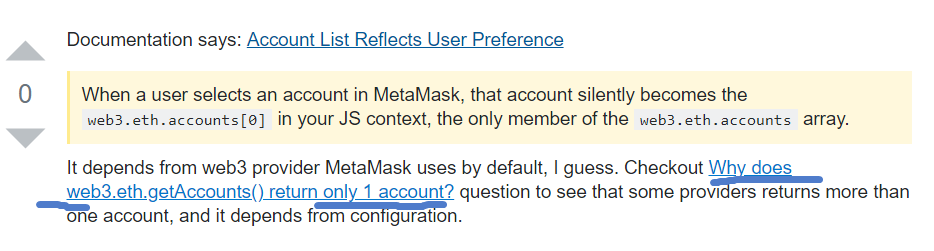


* + Note about getting all Ethereum accounts in case we use MetaMask

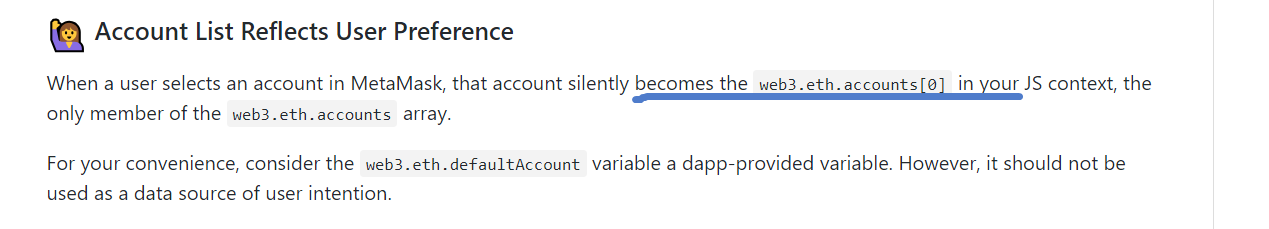
Here is what was asked on Stackoverflow <https://stackoverflow.com/questions/52976890/get-all-ethereum-accounts-from-metamask>



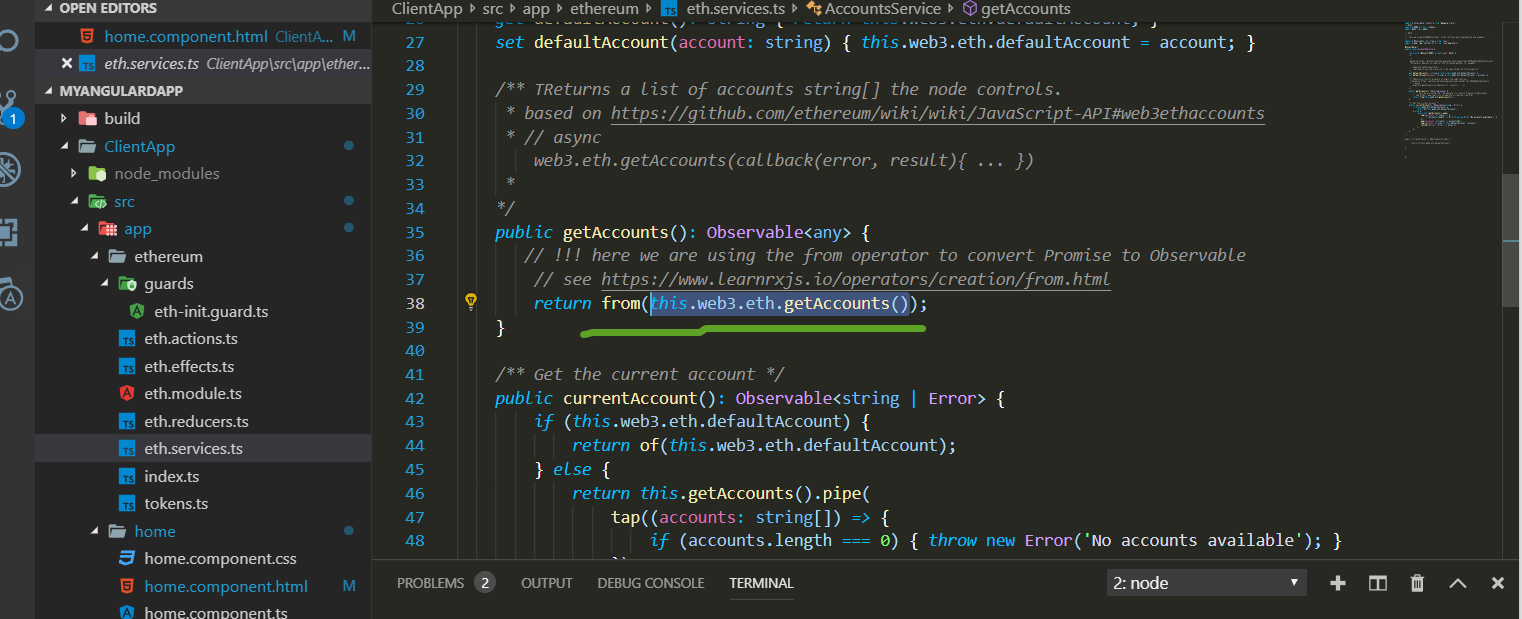
And here is the answer:

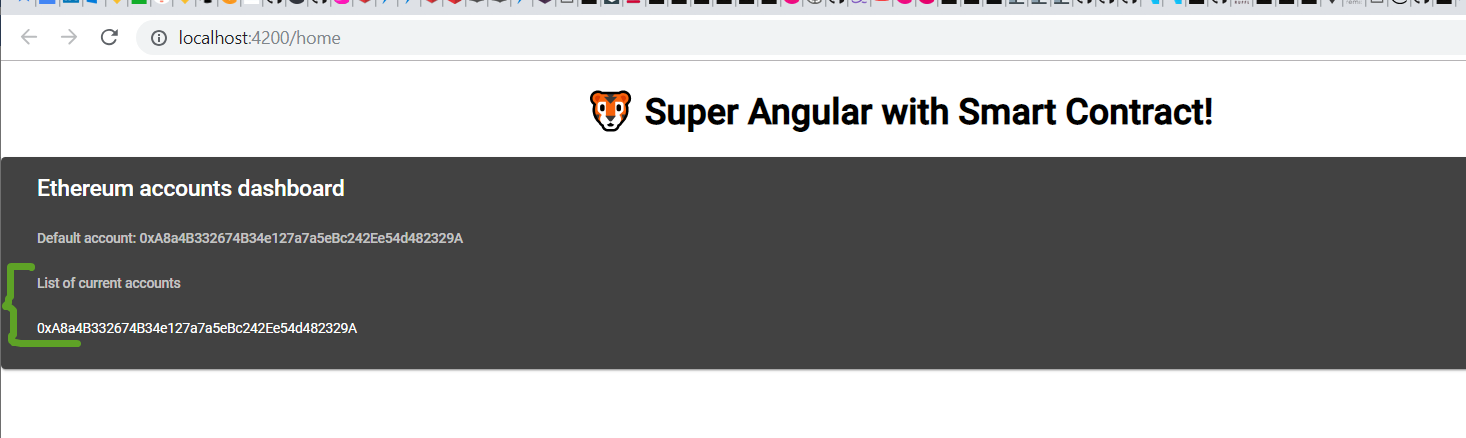


And here is the original source: <https://github.com/MetaMask/faq/blob/master/DEVELOPERS.md#raising_hand-account-list-reflects-user-preference>



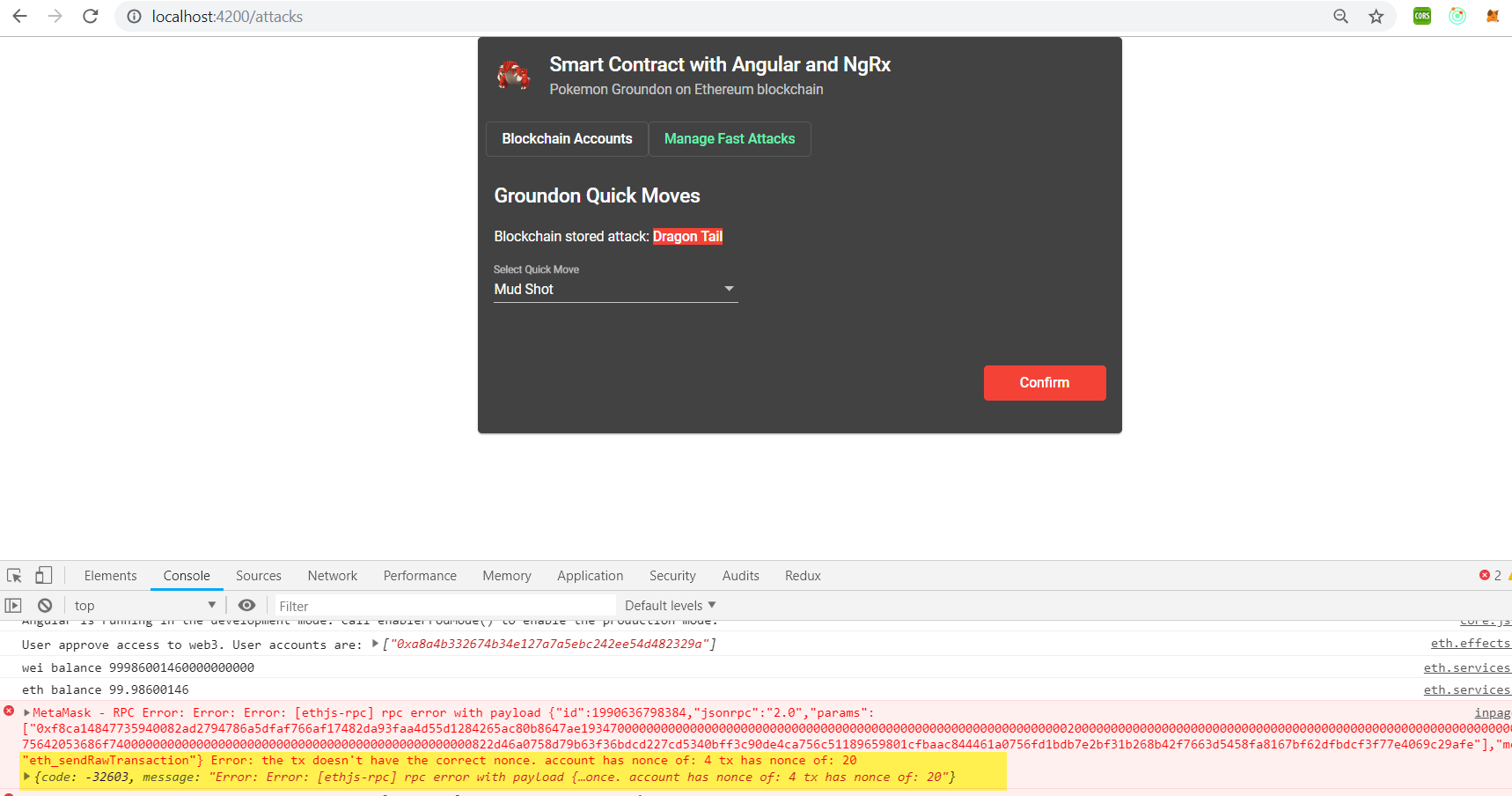
This explains why we see only one account returned from





Some stupid error

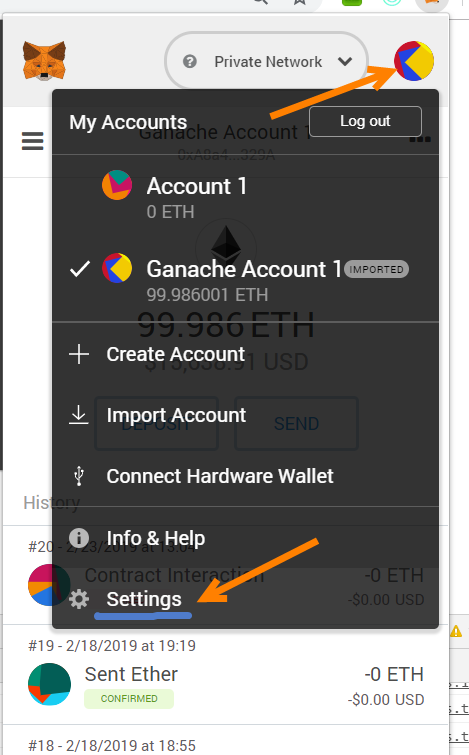
We could encounter this error sometime:



To fix it we need to reset MetaMask account:

(see <https://ethereum.stackexchange.com/questions/30921/tx-doesnt-have-the-correct-nonce-metamask>)

Go to settings



And click Reset

