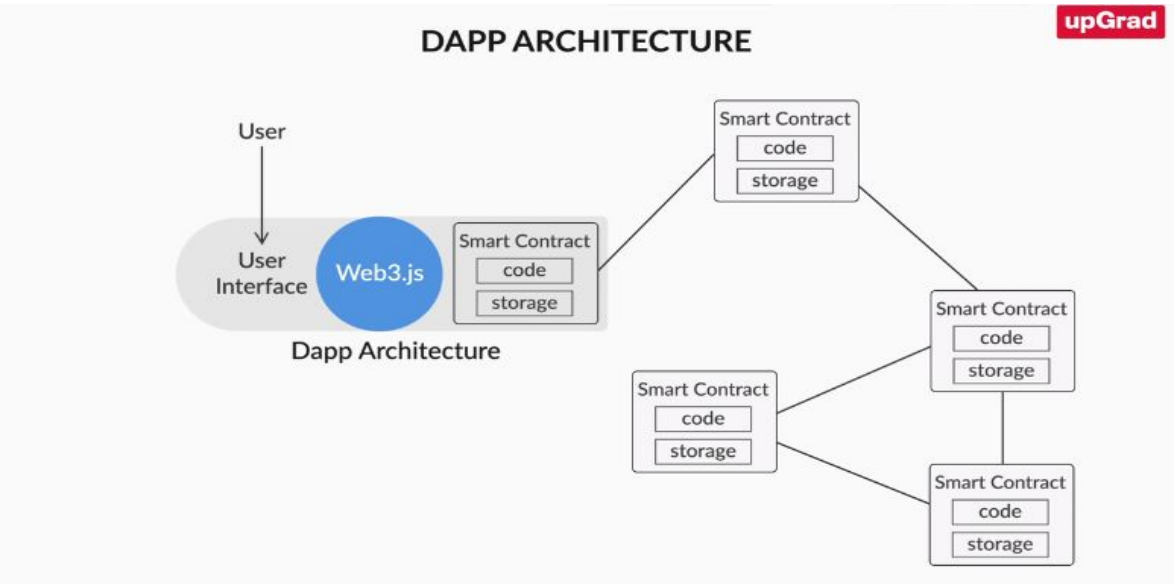
Lecture Notes

Smart Contract Development in Ethereum

Accessing Smart Contracts from the Front-end

After completing the backend part, the next part is to build a frontend application that end users will be able to use, and this application needs to find a way to communicate the smart contract.

DApp architecture



The frontend application needs to have some sort of process to be able to interact with one of the nodes in the entire Ethereum network. The network has several nodes, but the application needs to connect with only one node to be able to access, read and update any state of the blockchain on behalf of your application.

So, the Ethereum client with which you would connect needs to be decided by your front end application.
Once connected to the node, the following three elements should be passed to the application to execute a function deployed on the blockchain:

- Address of the smart contract
- Function that is to be called
- Input variables of that function

Web3.js is the only library that helps you connect the smart contracts to the front end. It is a JavaScript library that exposes a set of APIs that a frontend application can use to interact with the network client.
The network client is actually a Geth node, which understands the language.
JavaScript Object Notation Remote Procedure call (JSON RPC): This is a complicated language. So, we use Web3.js, which provides easy-to-use library functions over 'JSON RPC' that is understood by the Geth node.

You can use *web3* as an npm package for your *react* or *angular* projects; for basic HTML files, you can include *web3* as a script import statement. Once you have included this library, you will be able to access various functions as long as you are connected to a node and are accessing the correct function of smart contracts.

Web3.js

A Web3 **provider** is a particular client in the network that will be providing web3 services to import the interface to your smart contract.

After importing web3, you need to define the provider for which you need the IP address of the node that you want to connect to as well as the application binary interface (ABI).

ABI is a JSON representation of your entire smart contract. It is mostly hardcoded as a part of your frontend application. The IP address is also hardcoded inside your frontend application.

However, the process is quite risky for most consumer-facing frontend applications, as the IP address is hardcoded in your frontend applications. This could lead to instances wherein the Geth client becomes unavailable for a certain period. To overcome this problem, most production-level applications follow one of these two approaches given below.

1. They build a cluster of Geth nodes running in front of the load balancer, which ensures that the Geth service is available overall. However, following this approach is quite expensive.
2. Another approach is to use blockchain as service APIs. Companies provide a blockchain server or Geth clients as a service. One such popular service is INFURA, which allows you

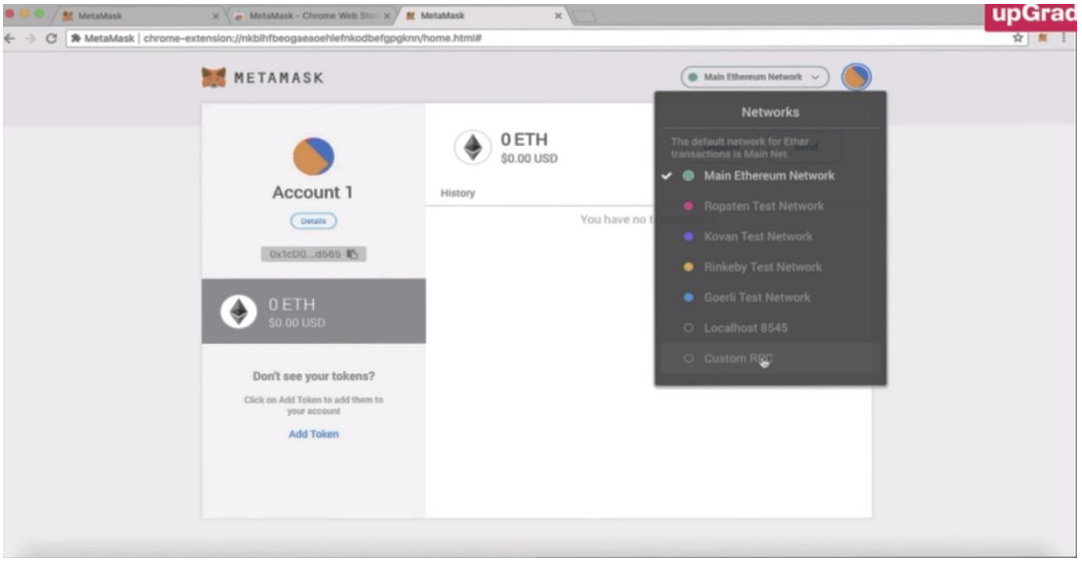
to connect to the blockchain using their API services and the Geth nodes that are available with them.

Another important challenge with web3 is faced when users are interacting with the front end application.
When transactions are sent from the front end applications, they need to be signed by the sender by providing a private key. Only signed transactions are accepted in the blockchain. So, every transaction needs to be signed, which leads to a poor user experience.

To overcome this problem, external services such as **Wallet** are used.

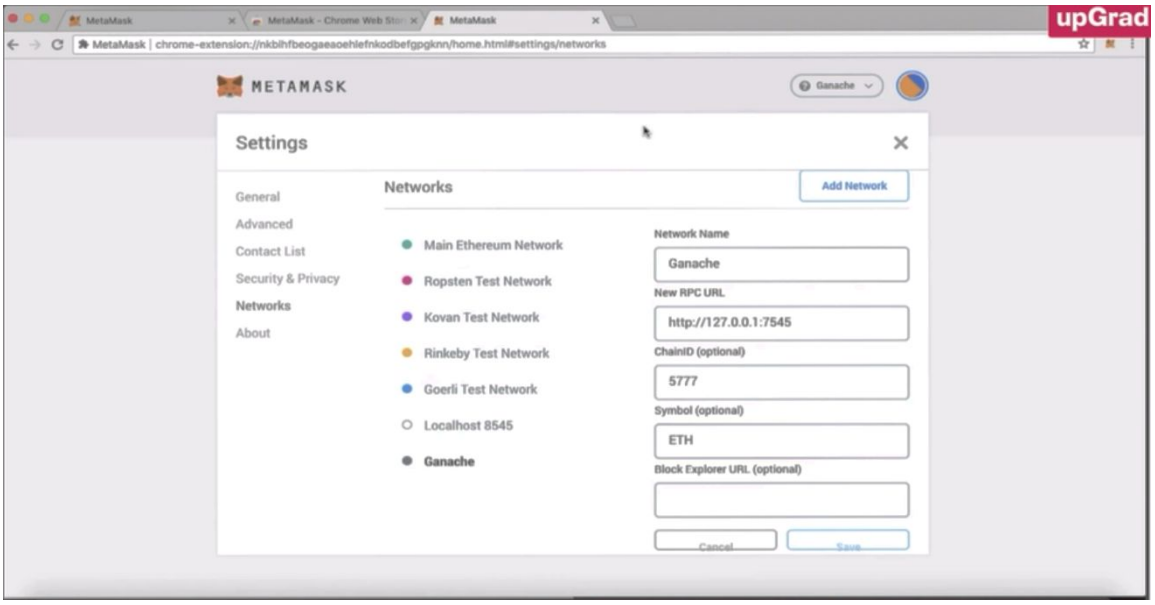
Connected this wallet to your Ganache instance

Step 1: Click on the 'main Ethereum network' option and then select custom RPC.

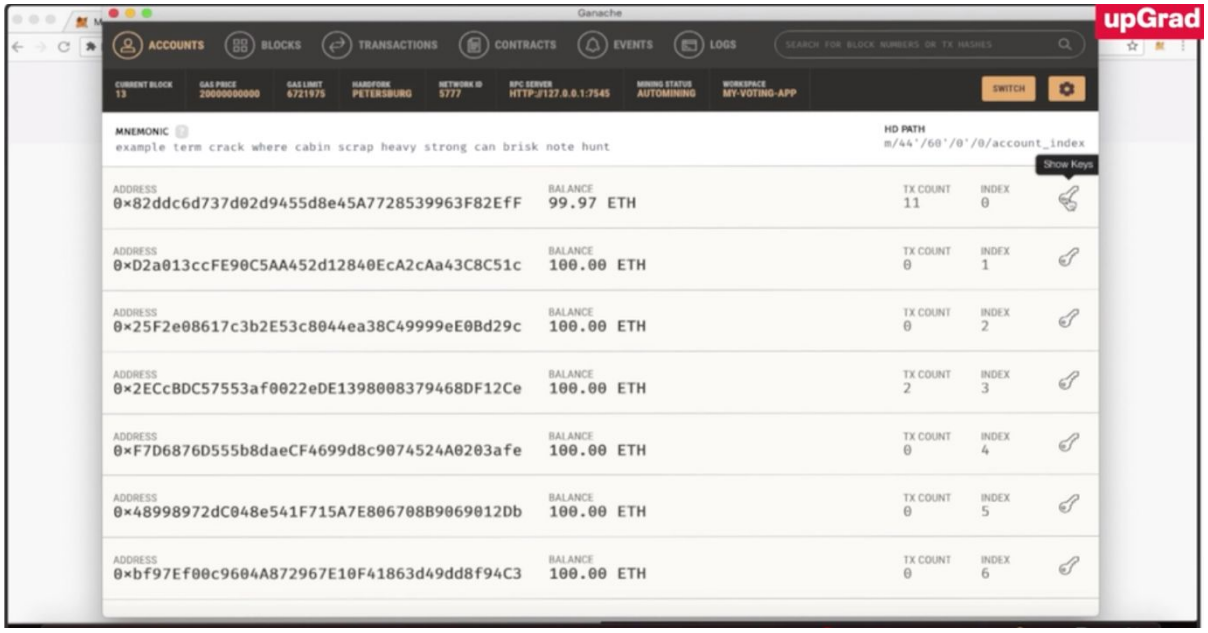


Step 2: Fill in the required details:

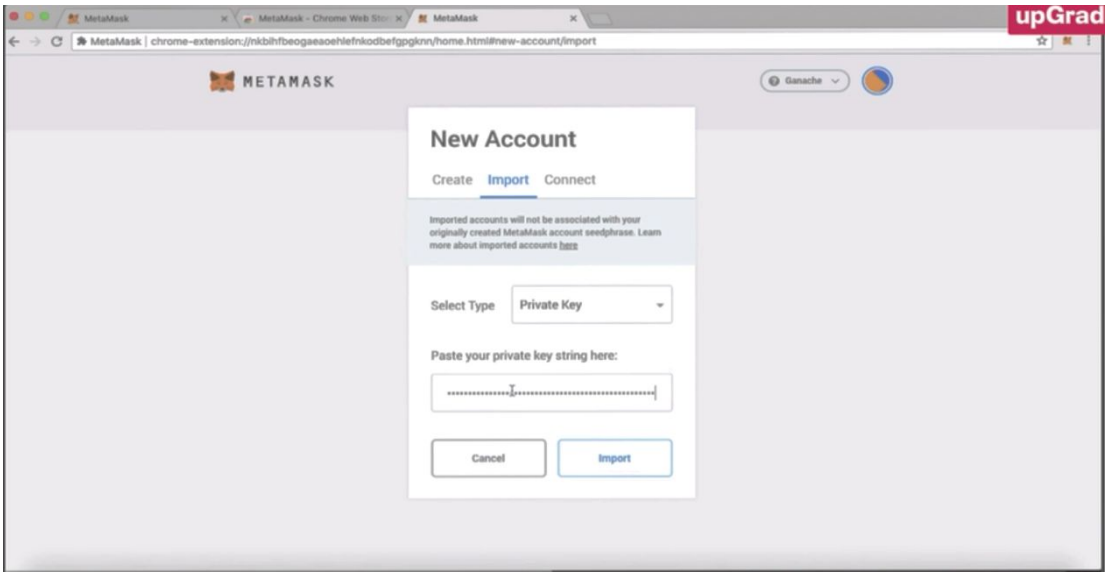
- Network Name: Ganache (it can be anything)
- RPC URL: http://127.0.0.1:7545 (same as the Ganache)
- Chain ID: 5777 (Value given in Ganache)
- Rest of the parameters are optional
- Save these settings
- Then, you will view your Ganache instance created in the network column



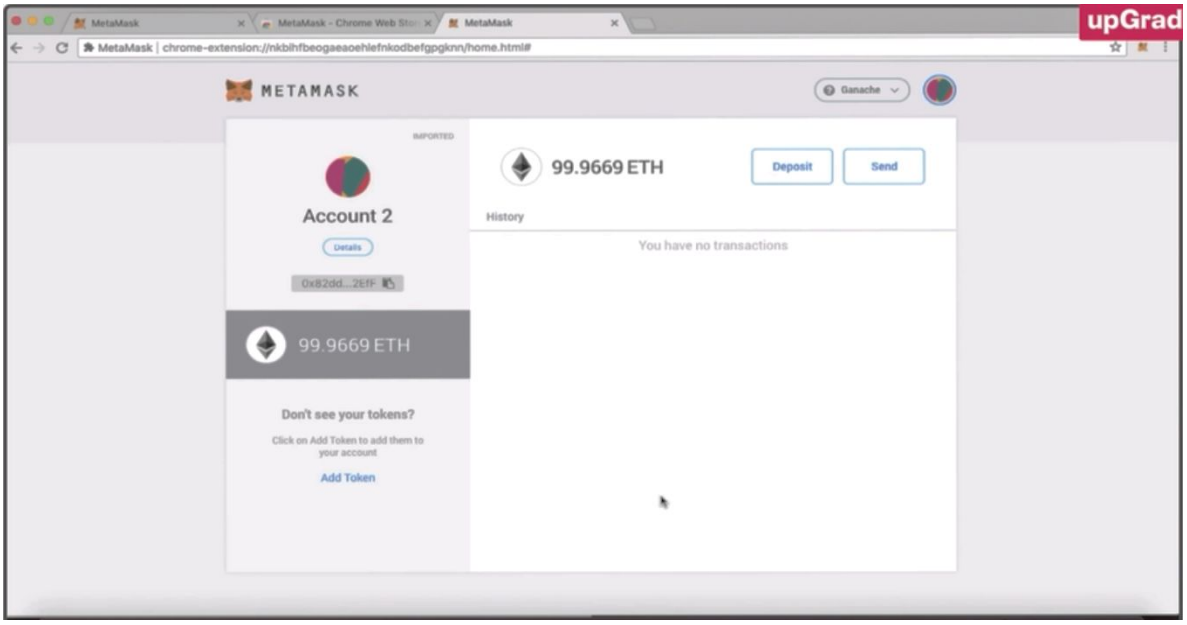
Import the account in your Metamask Wallet



Go to your Ganache instance and copy the private key of any account by clicking on the key sign on the right side of the accounts.



Now, paste that private key in the ‘New Account; page and click on ‘Import’.



Now, you will notice that the account is imported and connected with the Ganache network with the correct amount of ethers that you have in the Ganache instance.

The imported account can be used to sign transactions when you access any DApp on the internet that is using your metamask wallet as the provider to connect to the blockchain.

Example: Connect your smart contracts to the frontend (REACT) with Web3.js

You can use a variety of frameworks to develop the front end, such as **Angular**, **React** or **HTML**.

Now, you will learn about the steps involved if you opt for react.

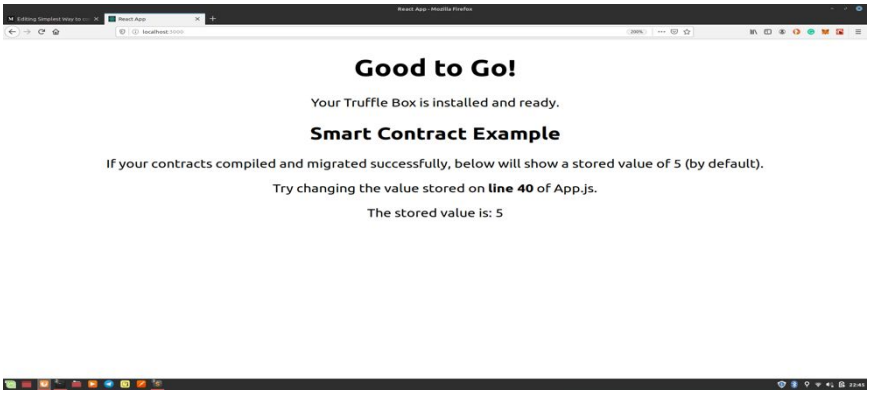
Steps:

- Create a directory.
- ‘cd’ to that dir.
- Run cmd – **truffle unbox react**.

- Ensure successful installation.
- **Edit Truffle-config.js:** Once you unbox the react component from truffle, the first job is to edit the **network object** in the **module.export** section. Set the ***port to 7545.***
- You need to connect your DApp to a local blockchain. In your case, the Ganache UI will do this. Most importantly, Ganache UI listens to port 7545, which is why you need to set your port to 7545 in the tuffle-config.js.
- Set up the Metamask and import one account from Ganache as explained in step 2 of Metamask section.
- Move to the directory where you unboxed the truffle react.
- Enter the command ***truffle compile***.
- Then, enter the command **truffle migrate**.

Note*: As soon as you migrate your smart contract, a small amount of ether will be deducted owing to gas. Now, click on the Metamask extension on your browser and check if some amount has been deducted. If yes, your set-up is completely fine, and you are good to go.*

- Go to the client directory and enter the command ***npm start***.
- Open the browser and go to localhost:3000.
- You will notice a Metamask pop-up.
- This is only a simple transaction that you need to approve to get the value from the smart contract to the front end part. Simply click on the confirm button.



You have successfully integrated React with your solidity smart contract with WEB3 and Ganache UI.

Example-2: Voting DApp

Refer to this link for Voting DApp.

References:

https://medium.com/coinmonks/simplest-way-to-connect-your-smart-contracts-to-the-front-end-react-with-web3-js-1e75702ea36a

https://developers.rsk.co/tutorials/frontend/first-frontend-web3-injected/

https://techbrij.com/web-ui-smart-contract-ethereum-dapp-part-4