VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

THE INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

ETHEREUM BLOCKCHAIN FOR REAL-ESTATE

Advisor: Dr. Tran Thanh Tung

Student name: Huynh Le Minh Thinh

A thesis submitted to the School of Computer Science and Engineering

in partial fulfilment of the requirements for the degree of

Bachelor of Computer Science

Ho Chi Minh city, Vietnam

2020

ETHEREUM BLOCKCHAIN FOR REAL-ESTATE

APPROVED BY:

_____

Tran Thanh Tung, Ph.D.

_____

_____

_____

_____

THESIS COMMITTEE

ACKNOWLEDGMENTS

I would like to express my very great appreciation to Dr. Tran Thanh Tung for his valuable and constructive suggestions and guidance during the planning and development of this thesis work. His constant encouragement and support helped me to achieve my goal.

In addition, I also appreciate teachers who have conveyed valuable knowledge for me during my study period.

LIST OF ABBREVIATIONS

ABI	Application Binary Interface
API	Application Programming Interface
CLI	Command Line Interface
DOM	Document Object Model
EOA	Externally Owned Account
ERD	Entity Relationship Diagram
EVM	Ethereum Virtual Machine
I/O	Input / Output
JSX	JavaScript Syntax Extension
NPM	Node Package Management
P2P	Peer-to-Peer
RPC	Remote Procedure Call
SPA	Single Page Application
UI	User Interface

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

1.1. Problem Statement

The government now wants to apply 4.0 technologies into properties' management. One of the most considered issue remained is trading real-estate properties. Since scamming in exchanging real-estate brings severe damage for traders, as scammers performing multiple transactions from a single house, to be more specific, selling a house many times.

Moreover, sellers always commit to pay rewards for buyers if some conditions have been reached. However, this need the verification of 3rd party, such as bank. This lead to the more extensive procedure. Buying a house is never easy !

1.2. Motivation

Thus, a proposed solution is using Ethereum Blockchain technology, which provided smart contracts to solve the above issues without interference of legal agents.

Since smart contracts only execute as they are programmed, this eliminate all of the third party involved, scamming issues and fake commitments from sellers.

1.3. Objective

The main goal of this thesis paper is to provide the fundamental knowledge of Ethereum technology, and show blockchain capabilities compared to traditional Web Application Development.

This paper contains system requirement, which shows some system criterion that must be fulfilled, then showing system architecture and system design. Last but not least, shows all implementation how to construct a demonstration for this thesis.

In this thesis, I'll create the first version of Ethereum Application in real-estate management aspect that perform all transaction using Ethereum blockchain platform.

1.4. Assumption

However, the usage of Blockchain in trading real properties is not yet widely and legally, the government still have not applied this house exchange system in the real world.

In conclusion, this thesis paper assumes all of house properties in blockchain system are legal and exchangeable as they are deployed by the sellers.

CHAPTER 2. BACKGROUND / APPLIED TECHNOLOGIES

2.1. Overview of web 2.0 technologies

A dynamic website, or web 2.0 with REST API is the most common web structure nowadays. The main advantage of a dynamic website is that developers can quickly update all data in an organized and standardized manner to build the different product pages or category pages according to user requirements. In conclusion, a dynamic web site must reach these following criteria:

- It must act like a functional website.
- It must be easy to update.
- Developers could be able to update the new content, helping to bring users back to the site and competing with search engines.

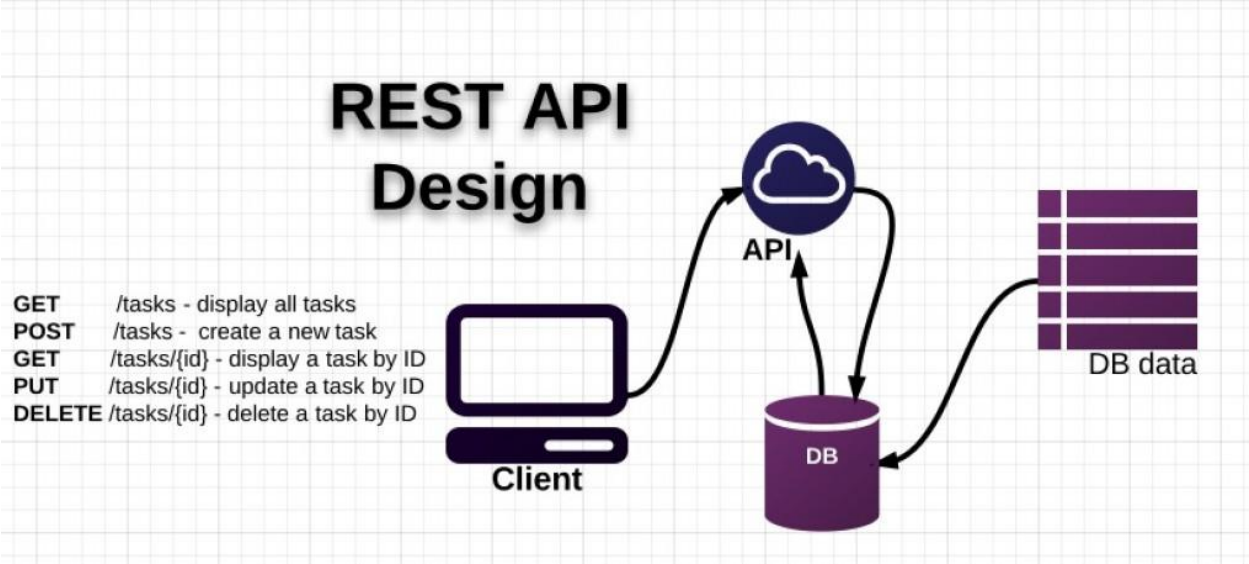

Figure 2.1. Web2.0 System with REST API [1]

2.1.1. NodeJS (Running Environment) and NPM

2.1.1.1. NodeJS

Node.js is a framework that builds on Google Chrome's JavaScript Engine (V8 Engine), or Chrome's JavaScript runtime to quickly build and easily scale network applications. Node.js uses an event driven model called non-blocking I/O [1] which makes it lightweight and powerful, perfectly suited for data-intensive real-time applications running across distributed devices.

Node.js is an open source, cross-platform runtime environment designed to build server-side and networking applications for JavaScript programming language. Applications Node.js are written in JavaScript and can run on different operating systems such as OS X, Microsoft Windows and Linux within Node.js runtime [2].

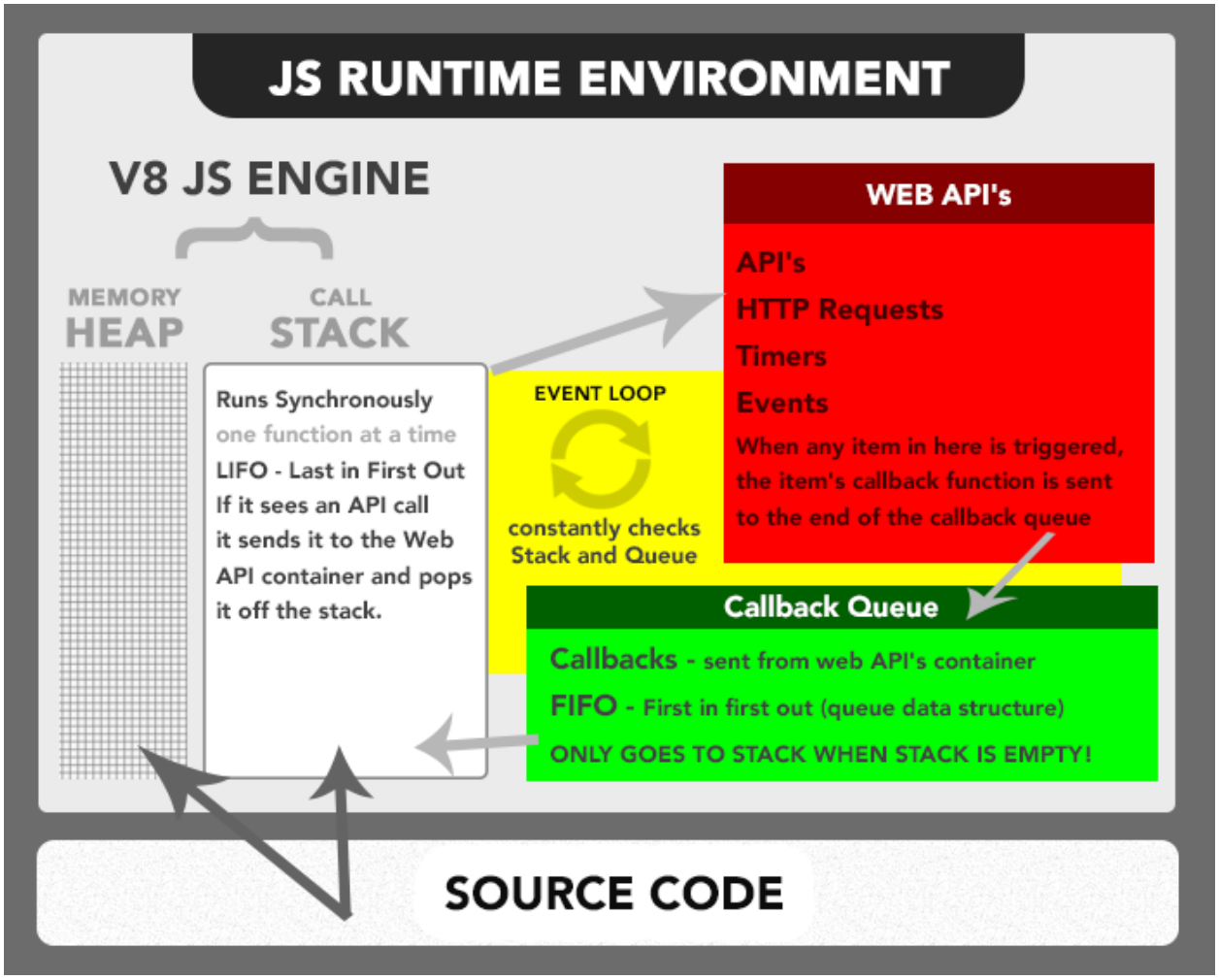

Figure 2.2. NodeJS Runtime Environment - Full Explanations [2]

Furthermore, Node.js provides a huge library that includes various JavaScript modules to greatly simplify web application creation using Node.js.

The below figure shows the introduction of JavaScript Engine in a fundamental perspective: Call Stack, Event Loop, and Task Queue [3].

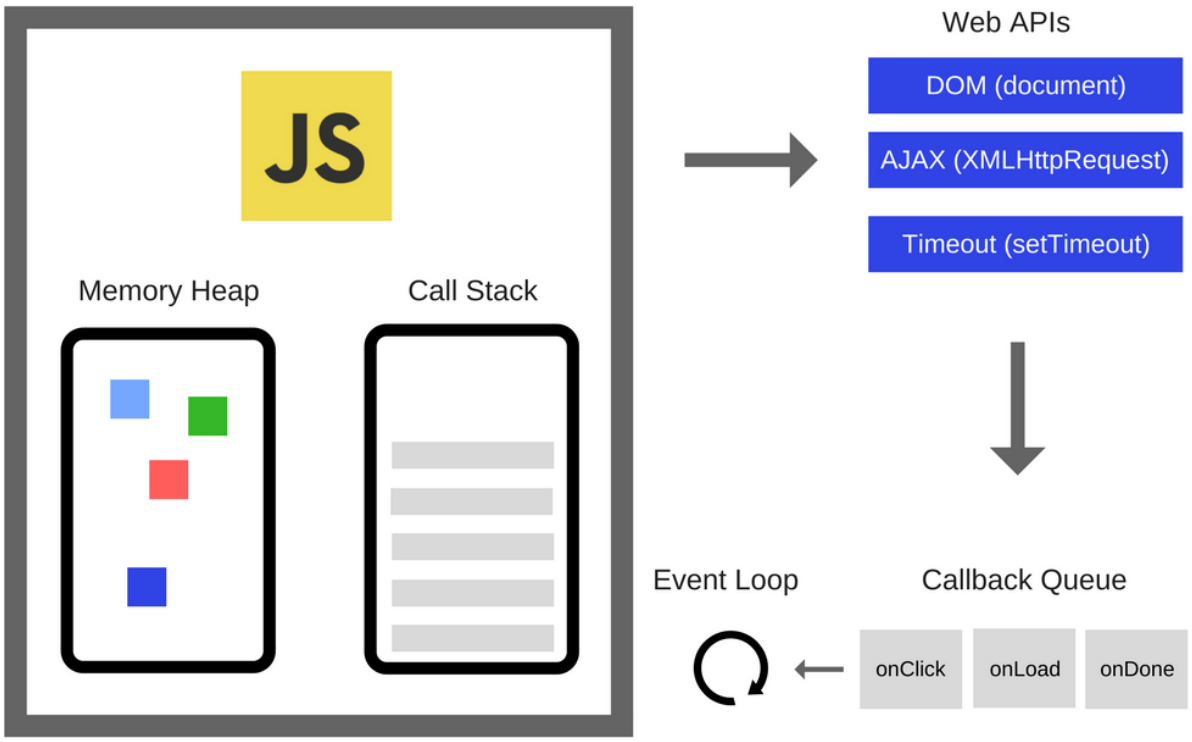

Figure 2.3. NodeJS Runtime Environment - Simple [3]

2.1.1.2. NPM

Installing Node.js comes along with NPM [4] which translates to "Node Package Manager". NPM is an online registry hosting packages for developers to download and managing those packages. Those dependencies are added locally on your computer so that you can use them whenever needed on your project.

After installation of Node.js, tracking your project package dependencies requires creating a simple file named package.json at the current path folder, then run the following command <npm

init> and answering the prompted questions or <npm init --yes> to immediately initialize the empty package.json file.

There are 2 main ways to use NPM, which is installing the packages globally or locally. The main difference between local and global packages is this:

- Local packages are installed in the directory where you run npm install <package-name>, and they are put in the node_modules folder under this directory.
- Global packages are all put in a single place in your system (exactly where depends on your setup), regardless of where you run npm install -g <package-name>.

2.1.2. ReactJS with Babel and Webpack (Front-End / Client Side)

2.1.2.1. ReactJS

React is a JavaScript library primarily meant for the hairy problem of measuring **state**.

React is "declarative", in that the main goal of your logic is telling the program "it should look like this" as opposed to "imperative", which is more like saying "you should do this".

Let's say I want to update the contents of an element based on a user clicking a button.

- **Vanilla JS (Imperative)**

```
button.onclick = function(e){
  document.getElementById("target-element").innerHTML = e.target.dataset.text
}
```

Figure 2.4. Click Button with Vanilla JS

❖ **React JS (Declarative)**

```
handleClick(e){
  this.setState({text: "NEW TEXT OR WHATEVER"})
}

render() {
  const results = this.state.activeTerms.map((term) => {
    return <div>{term}</div>
  })
  return (
    <div>
      <button onClick={this.handleClick}>Click Me</button>
      <div id="target-element">
        {this.state.text}
      </div>
    </div>
  );
}
```

Figure 2.5. Click Button with React JS

In React HTML is always re-rendered when the state is changed so the app always has a concept of what "text" is. In plain JS, you explicitly tell the component what its text should be. This example itself is probably easier to do in plain JS (or jQuery), but as an interface grows in complexity it pays dividends to follow this way of handling state.

2.1.2.2. Babel

Babel is a JavaScript Transpiler !

Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript, which helps the current and older browsers or environments to run the new standard of JavaScript code. Babel could do the following functions:

- Transforming JavaScript syntax.
- Polyfill features that are missing in your target environment (through @babel/polyfill).
- Source code transformations (codemods).

```javascript
// Babel Input: ES2015 arrow function
[1, 2, 3].map((n) => n + 1);

// Babel Output: ES5 equivalent
[1, 2, 3].map(function(n) {
  return n + 1;
});
```

Figure 2.6. Babel Transpiler I/O [4]

2.1.2.3. Webpack

At its core, webpack is a static module bundler for modern JavaScript applications. It takes disparate dependencies, creates modules for them and bundles the entire network up into manageable output files. This is especially useful for Single Page Applications (SPAs), which is the standard web page for Web Applications today.

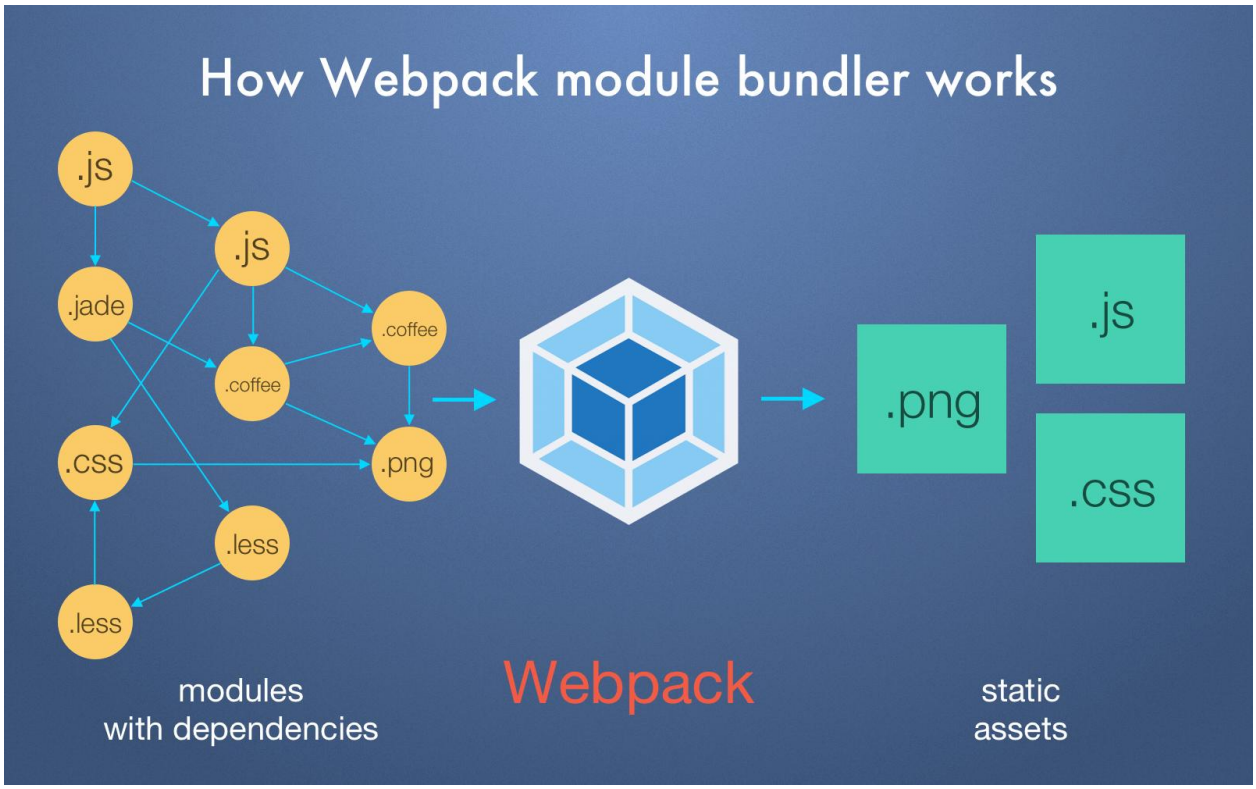

Figure 2.7. How Webpack Works [5]

2.1.3. ExpressJS with Nodemon (Back-End / Server Side)

2.1.3.1. ExpressJS

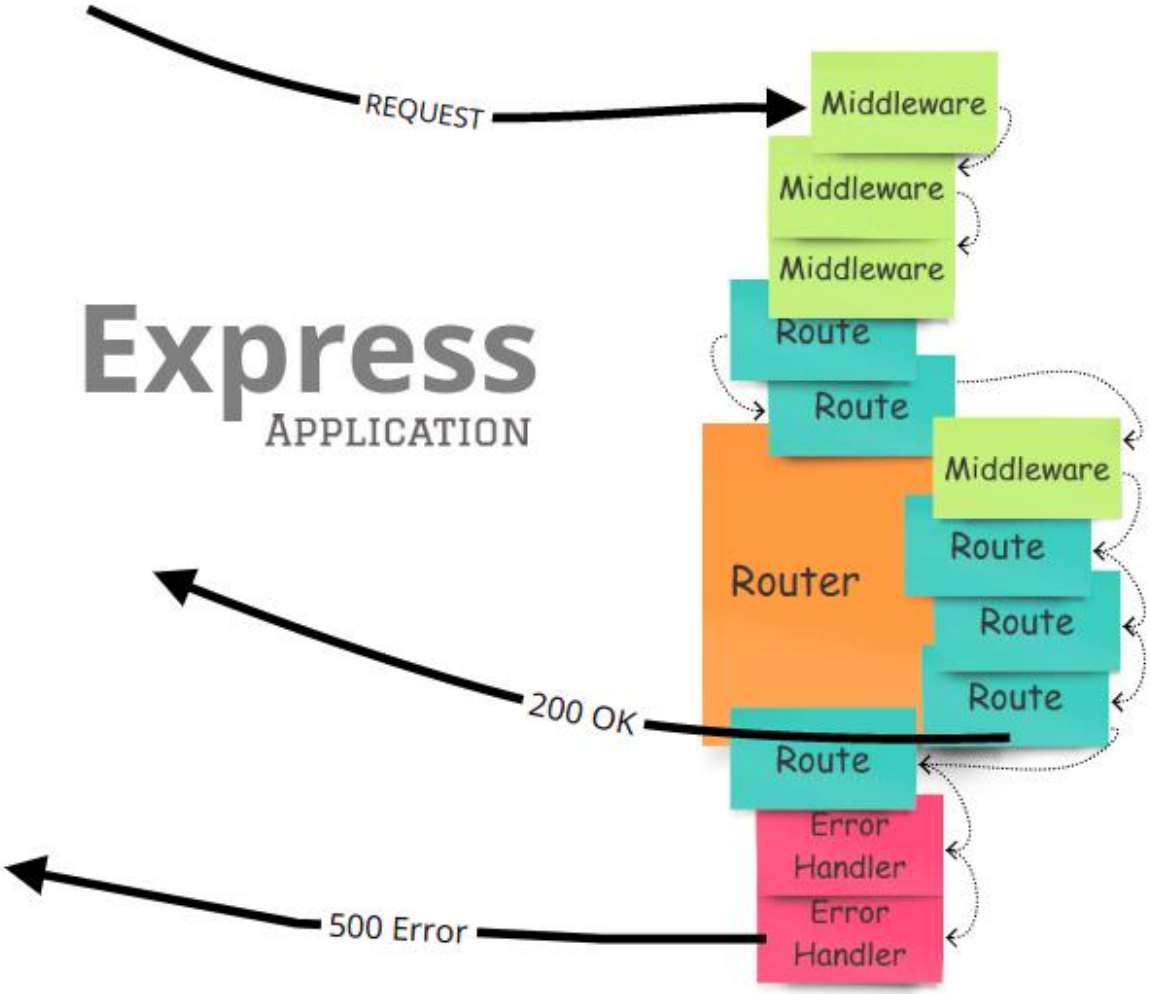

Figure 2.8. Introduction to ExpressJS [6]

ExpressJS is the most popular web framework based on NodeJS platform, it is also underlying as a standard library for numerous of other popular web frameworks. ExpressJS provides the following mechanisms for developers, such as:

- Write handlers (routers and services) for client-side requests with different HTTP at different URL paths (routes).

- Integrating with "view" engine that helps rendering a HTML template in order to generate responses by inserting data into templates.
- Setup common web application settings, such as port's configuration used for connection and the path folder contains HTML templates that are used for rendering the response.
- Add additional "middlewares" at any point within the request handling pipeline to process the incoming requests.

Although ExpressJS itself is fairly lightweight, developers have built useful and usable middleware packages to solve nearly any web development problem. For example, Express supports various libraries such as cookies, sessions, user logins, URL parameters, POST data, security headers, and many more. Developers could easily find a list of middleware packages on Express Middlewares maintained by the Express team (along with a list of some common 3rd party packages).

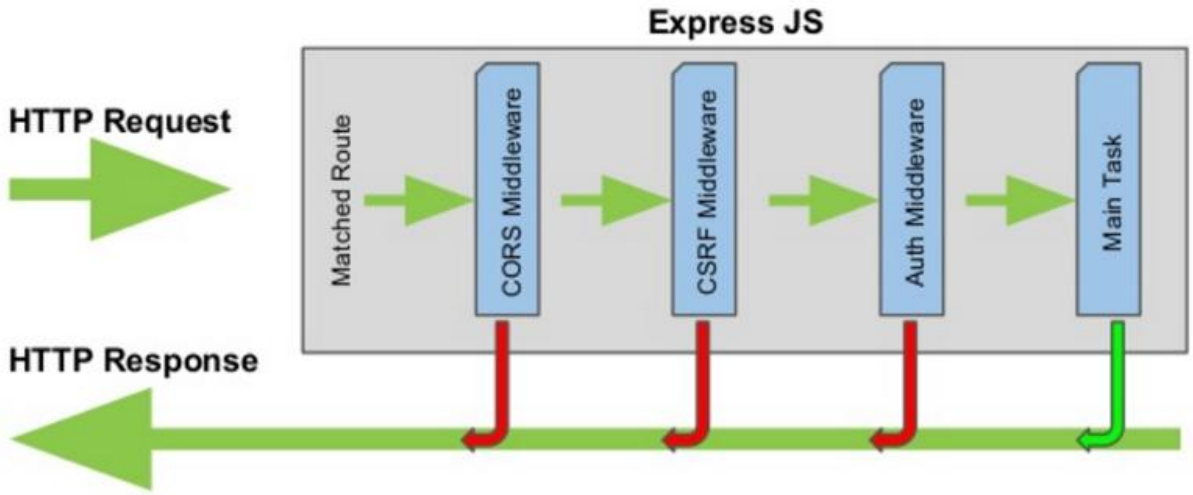

Figure 2.9. ExpressJS Workflow [7]

2.1.3.2. Nodemon

Nodemon is a tool that makes it easier than ever before to build NodeJS-based applications by automatically restarting the Node application whenever any file changes are detected in the directory.

Nodemon does not need any further changes to your code or creation process because it is a node-only wrapper. Nodemon could be used when executing the script by replacing the word

node in the command line with nodemon. The usage of Nodemon is specified as the following command <nodemon script.js>.

2.2. Overview of Ethereum Blockchain technologies

2.2.1. Blockchain' fundamental background

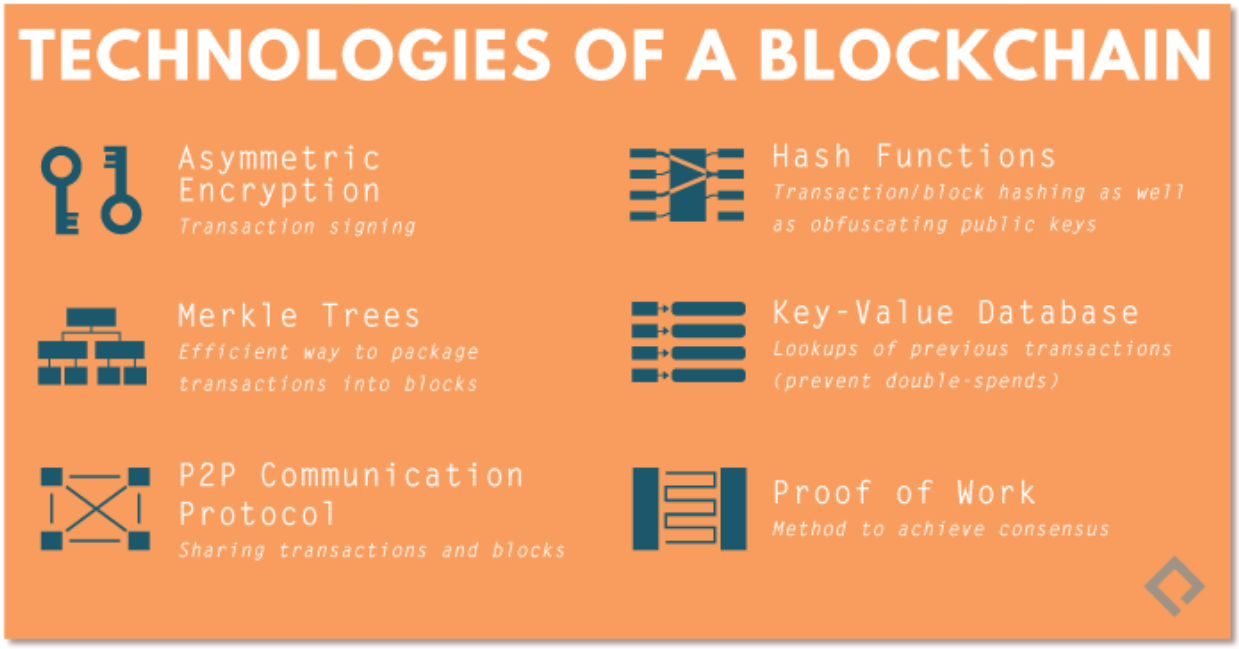

Figure 2.10. Fundamental Technologies of Blockchain [8]

Blockchain' precursor was authentically described in 1991 and its goal was to marking digital documents with timestamp in order to prevent contents from being tampered or to set the signed day before it was written.

The most fundamental theorem of Blockchain is Hashing and Merkle Tree.

2.2.1.1. Hash Function

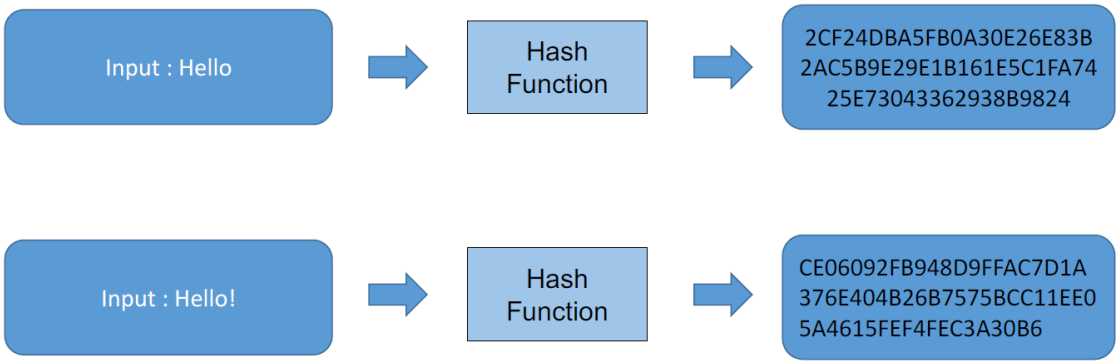

Figure 2.11. Hash Function [9]

Hashing is a one-way function, which takes input and produces a fixed size output, that cannot be reversed after running. Moreover, if the data in type of binary is changed 1 bit only, the result would be significantly replaced. Last but not least, there is no case to figure out the input if we only have output string.

This cryptographic hashing must satisfy 3 properties:

- Collision-free: There's almost no case that 2 different inputs will produce the same output
- Hiding: It's hard to find out what's the input even if the output has been published.
- Puzzle-friendly: The only way to match the output is to randomly use as much as possible input.

2.2.1.2. Merkle Tree

Merkle Tree is a hash tree with multiple layer of blocks, which represents a type of data structure. And each block on the upper layers will be constructed by combining 2 blocks from the lower layers.

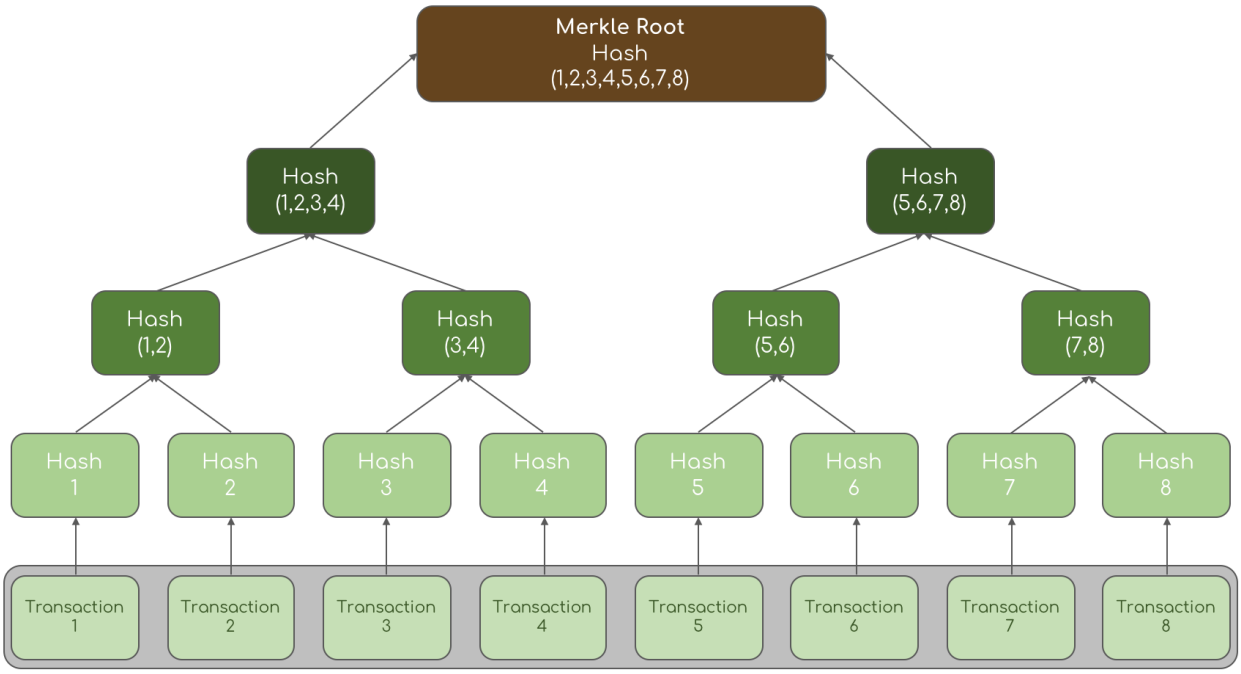

Figure 2.12. Transactions Merkle Tree [10]

Constructing Merkle Tree requires hashing all the data block, resulted in all different string of hashed values. Then each two adjacent of hashed value are hashed once again to make the hash value for the higher layer. Keep iterating until reaching the top of the tree, where the number hashed values in the highest layer is equal to 1.

The root hash will verify all the data blocks in the lowest layer, since a small change of data in any block will cause different hashed value on root hash.

Into details, a Blockchain is a distributed computing architecture, which uses peer-to-peer network protocol. Anyone is allowed to join and became a node of the network. When someone runs as a full-node, they must gets the full copy of the Blockchain' data, specifically in this case which is the main ledger. By this protocol, every node executes and records similar transactions to keep the Blockchain holds consensus mechanism across the whole network. Any interactions with Blockchain are resulted in adding transactions into a block. At a time, only one block can be generated and every block contains a hash of the previous block, which makes the chain become unbreakable.

2.2.1.3. Block's Structure

Each block included 3 main parts, which are header, transaction count and block content.

- **Header**: The block header is hashed twice to establish the fingerprint to which the next block refers.
  - *Technical data*: Included a magic identifier, version number (to define the set of protocol rules to which a block conforms) and block size.
  - *Merkel Root*: Distills all block transactions into a single hash value (the root of Merkel Tree transactions).
  - *Timestamp*: An approximate timestamp of a block when it was created. Timestamp is used in order to re-target the mining difficulty, if the network is making blocks too quickly or too slowly, for example.
  - *Previous block hash*: 2 times SHA256 hashing of previous block header (excluding magic ID & block size). This is the link that forms the blockchain.
  - *Difficulty target*: The value related to mining mechanism, which show how hard it is successfully mine the block. Also known as the difficulty of solving the cryptographic puzzle.
  - *Nonce*: A random number, which is the only thing miner may alter while mining to create different hashes until a suitable hash value is found.
- **Transaction count**: How many transactions are in the block.
- **Block content**: Holding a list of transactions.

The Figure 2.13 below illustrates the detail information of a Bitcoin Block.

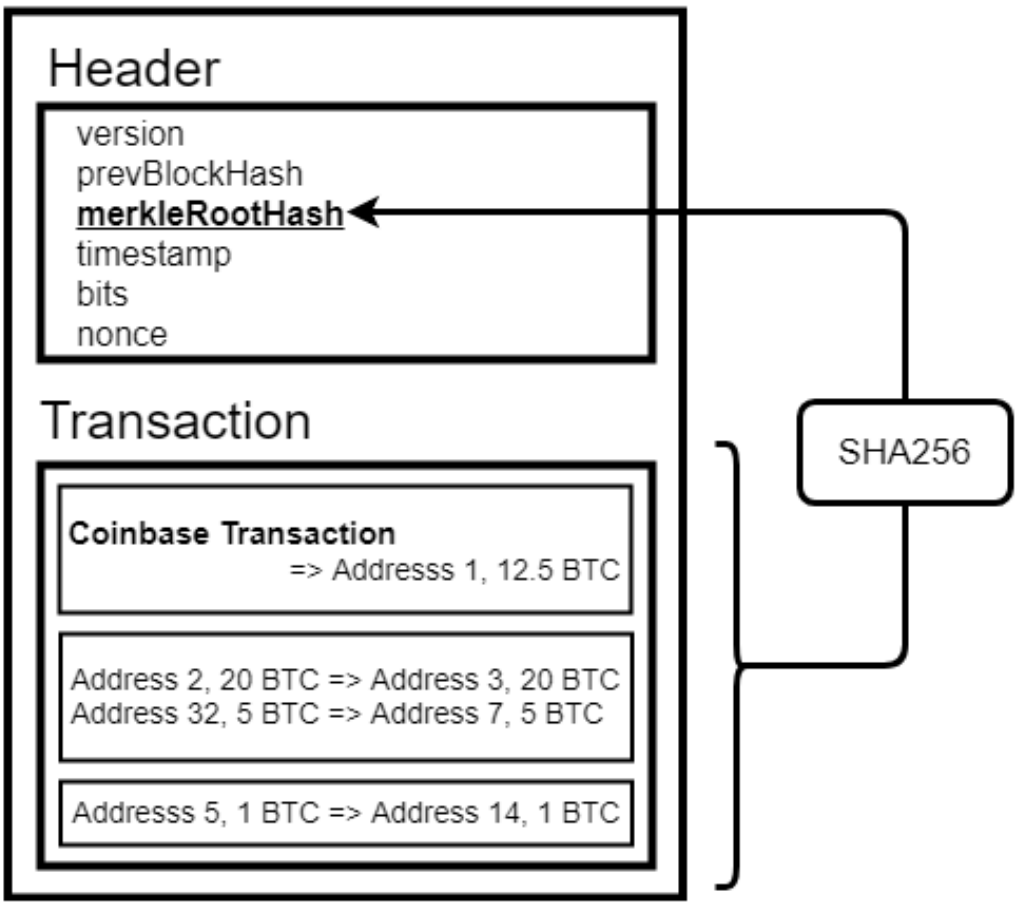

Figure 2.13. Semantic Bitcoin Block Components [11]

2.2.1.4. Blockchain' structure and Mining mechanism

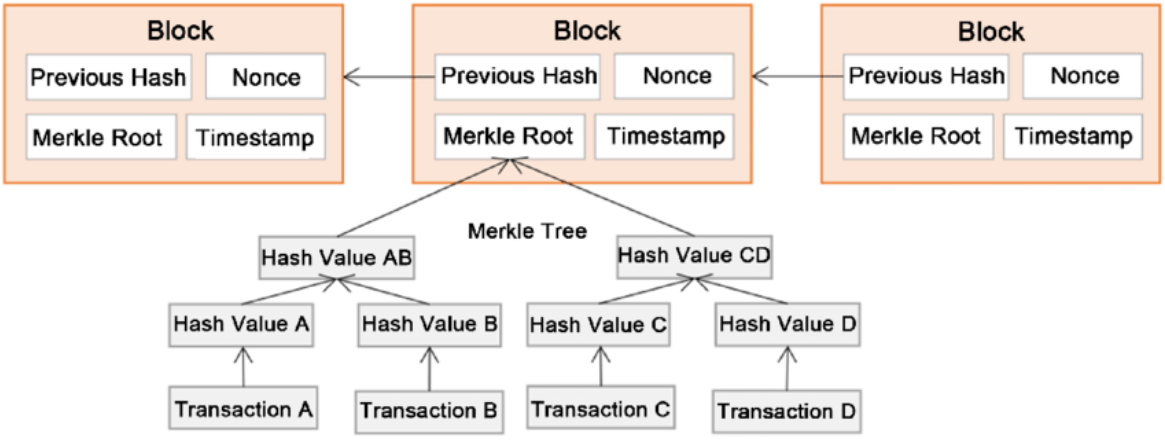

Figure 2.14. Bitcoin Blockchain Structure [12]

To create a new block and add it to a chain, Blockchain network has its own workers called miner. A miner start constructing a candidate block by gathering the transactions in the "*transaction pool*", a pool of new or unconfirmed transactions. This candidate block arrangement is same to a normal block structure. See Figure 2.15 below.

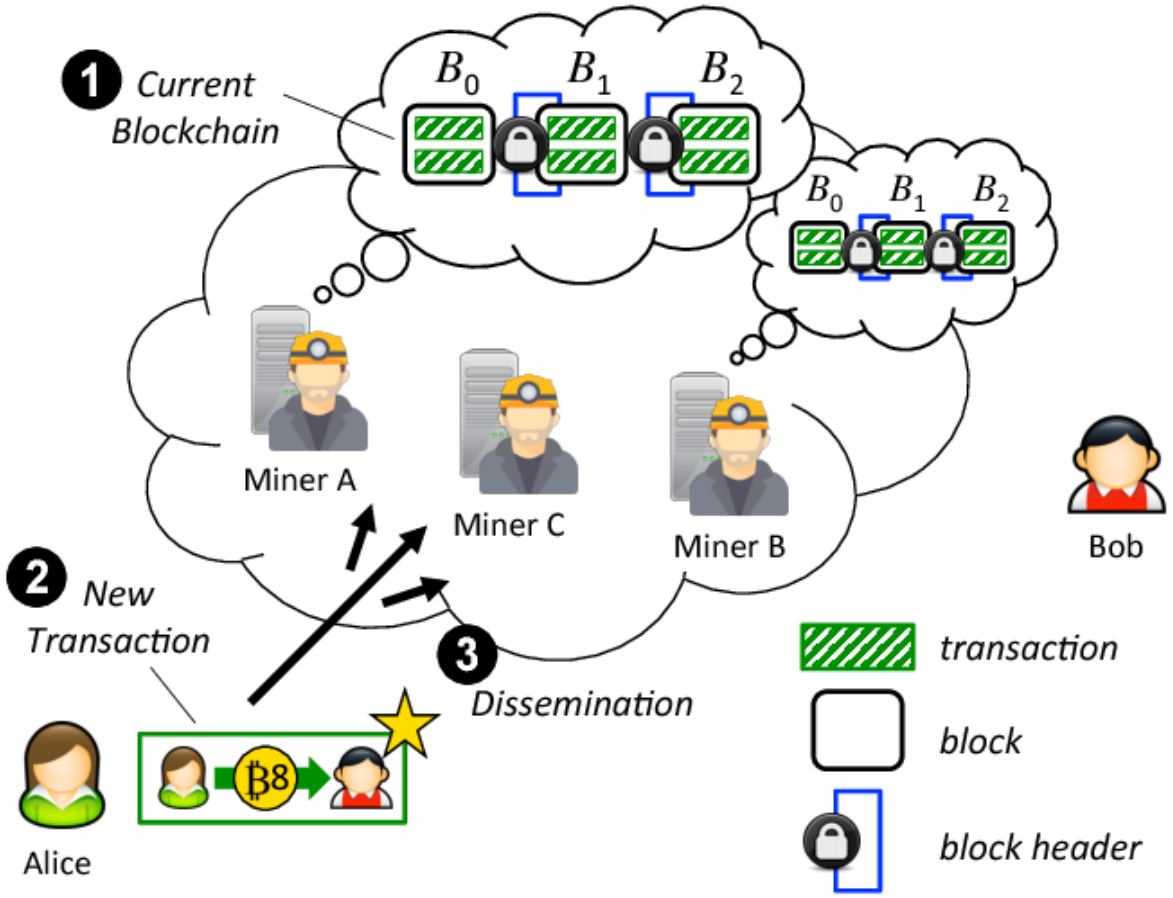

Figure 2.15. Miners Collect Transactions [13]

Then, miner spends resources (dedicate hardware and electricity) to compute the hash of the block's header. If the output is fulfil the *Difficulty target*, new block is created and linked to the top of the chain. Otherwise, that miner has to change the *Nonce* until an appropriated hash output is found. By this way, miner is "mining" for a new block. Since the computational process of solving for the solution to create a new block is costly and time-consuming, the output is literally called Proof-of-work. See Figure 2.16 below.

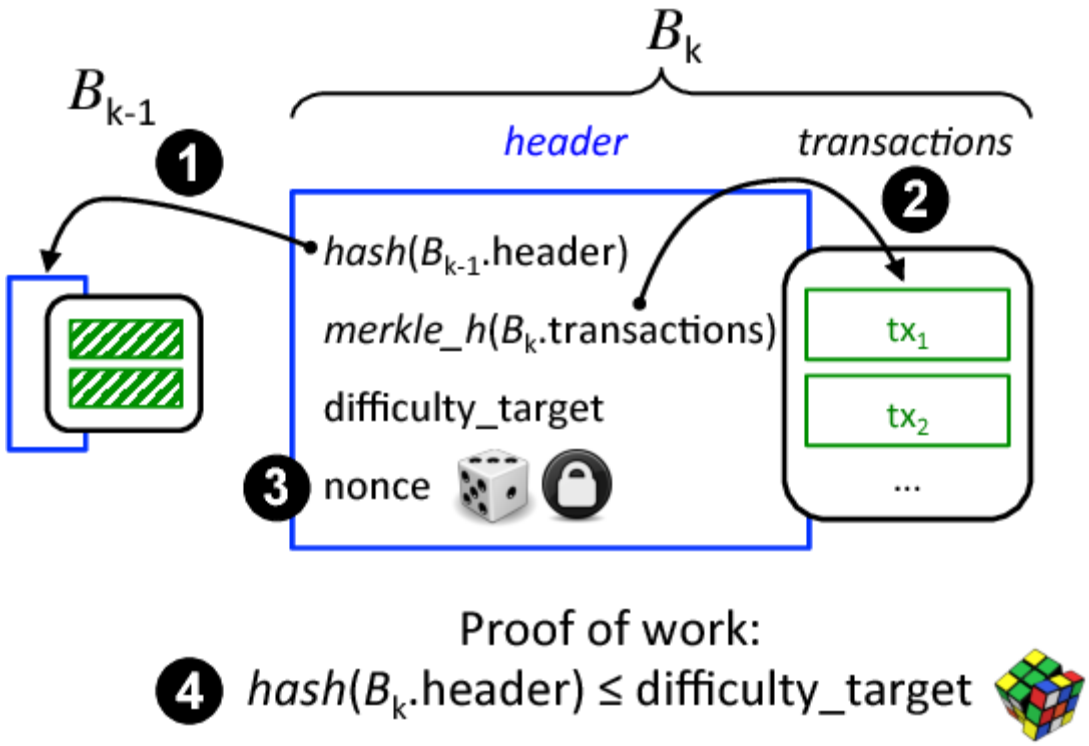

Figure 2.16. Proof-of-work Mechanism [13]

After a mining process is done, that new block is sent to every nodes on the network. Each node then verifies the block to make sure it is valid by hashing again the block data along with the miner's nonce to check the appropriated Difficulty target. After that, all the node in this network create a consensus protocol. They make agreement about which blocks are valid and which are not by following those criterion:

- Block header hash is appropriated to the block target.
- Block size is within acceptable limits.
- Block timestamp is less than a *T* time in the future.
- All transactions within the blocks are valid (also have a checklist on their own).

After successfully mining a new block, miner get a total of reward for mining block and transactions fees. See Figure 2.17 below.

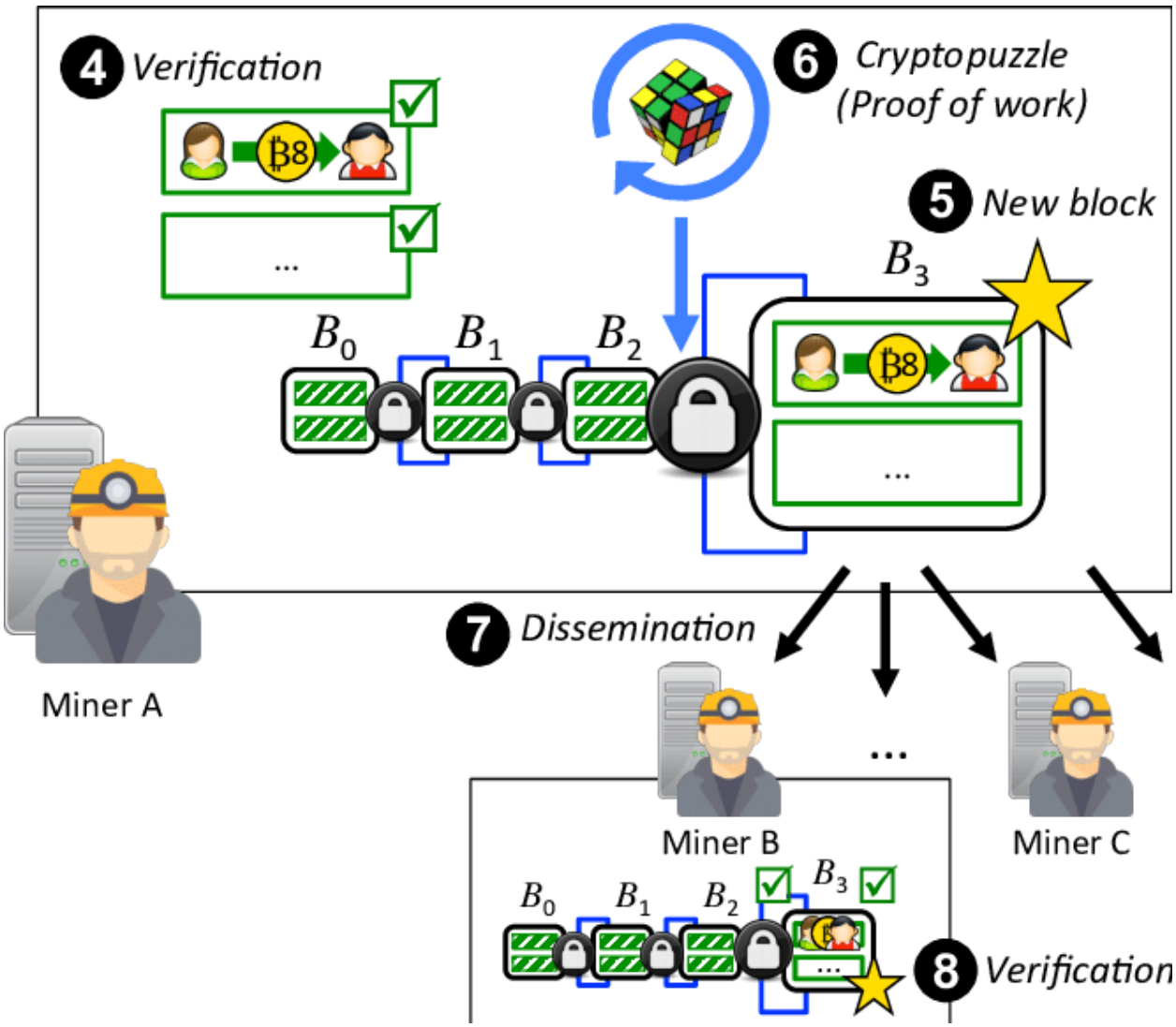

Figure 2.17. Blockchain Mining Mechanism [13] [14]

2.2.2. Asymmetric Key Pair

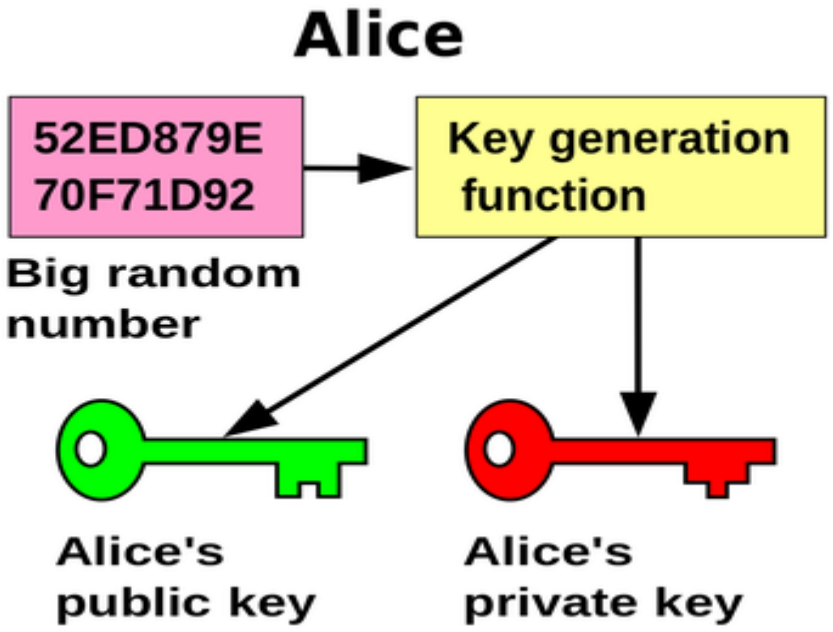

Figure 2.18. Abstract Keypair Generator [15]

2.2.2.1. Cryptographic Proof-of-identity

Cryptographic Proof of identity in simply means proving one's identity without revealing it. It use the cryptography techniques to generate a digital signature. This will make a person become anonymous since they does not need providing personal information, but still can prove the privileged possession of a property.

Here is what a cryptographic system can do. It generate a key pair, one is public key and another one is called private key. They are a set of long characters that are mathematically connected, which are large prime number based. A public key could be public like a username, and the private key must be kept secretly as your password. Once a property of a person is encrypted with his or her public-private key pair by the cryptographic algorithm, only that person can prove the ownership of that property. Since, it would take more than 1000's of years to break the cryptographic algorithm because of the computational limits nowadays. In the future, quantum computers might be a concern, but hopefully developers could find a way to deal with that technology.

Figure 2.19 below is an example of using key pair in cryptographic proof-of-identity.

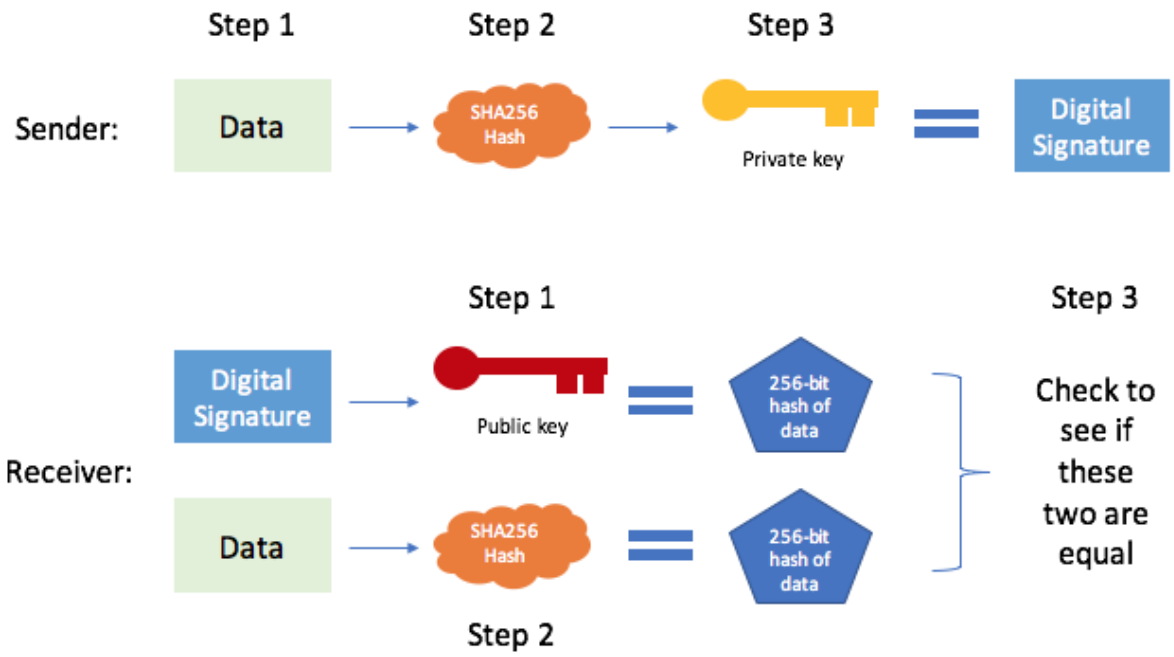

Figure 2.19. Asymmetric Keypair with Proof-of-identity [16]

2.2.2.2. Asymmetric Encryption / Decryption

Figure 2.20 below is an example of using key pair in cryptographic proof-of-identity.

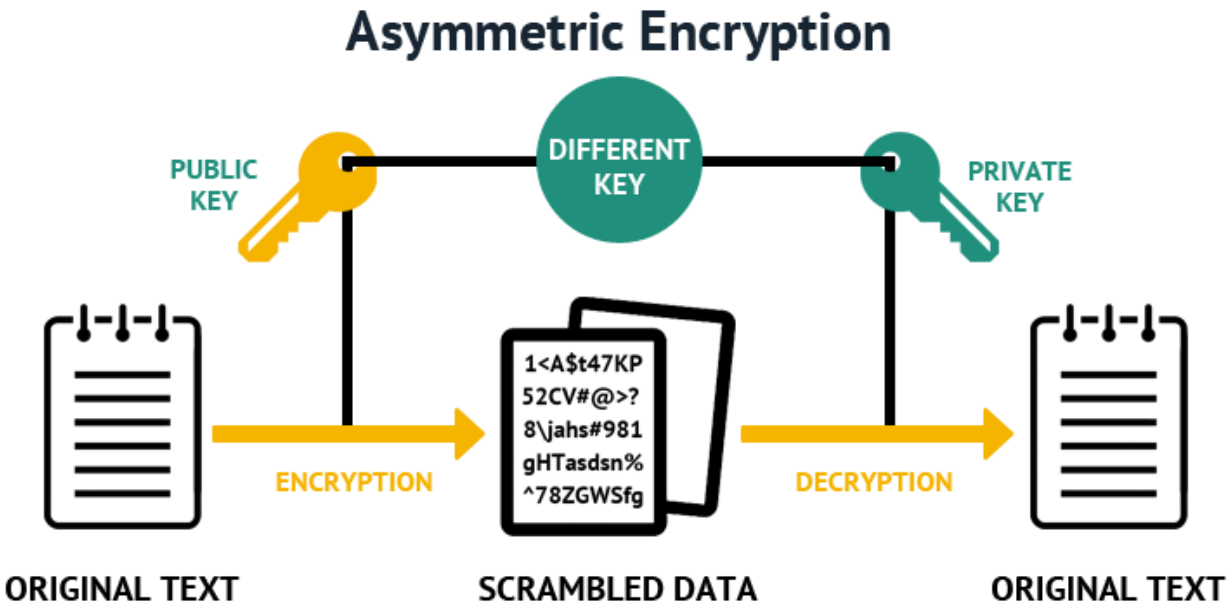

Figure 2.20. Asymmetric Keypair with Encryption and Decryption [17]

Rivest–Shamir–Adleman (RSA) — an asymmetric cryptographic algorithm, basically uses two different keys to Encrypt and Decrypt the Data. Public Key is used to encrypt while Private key to decrypt.

2.2.3. Peer-to-peer Network

2.2.3.1. Introduction to P2P Network

In simplest terms, a peer to peer network is two or more computing devices connected to each other to accomplish a task or set of tasks.

A P2P network can be an ad-hoc network where two computers are connected by USB to transfer files. It can also be a permanent network where all the computers in an organization are connected.

Every connected computer or computing device in a P2P network is called a node and serves as a client and server at the same time. There is no presence of a centralized server.

2.2.3.2. How P2P Network Works

Whenever a user wishes to download a file hosted on a P2P network, he will need to join the network via a client application called as “leech”. This client will then search for the file on the network called as “seeds”.

2.2.3.3. Examples of P2P Network

- Bit Torrent – Needs no introduction, this is the most popular peer-to-peer.
- DistriBute – World’s first P2P desktop deployment product for business use.
- Tor – Application that shields user information by sending traffic through a P2P network of relays set up by volunteers across the world.

2.2.3.4. Comparison to Server Based Network

Table 2.1. Server-Based compared to Peer-to-Peer Network Model

	Server-Based Network	Peer-to-Peer Network
Description	A server-based network has a centralized server that stores, manages, and communicates data from one	P2P network does not have any servers. Data management happens on the computers or peers in the

	computer to another. Users use local systems/computers to access, create, modify data which are stored on a server present at a remote location.	network.
Cost	Setting up a server based network require larger teams, complex architecture as compared to a P2P network	
Scope and magnitude of use	Server based network are pre-dominantly used for large organizations that handle and process complex data every hour, round the year	P2P networks are suitable for smaller organizations with dozens of employees performing relatively easier or less complex tasks like sharing files, printer access, etc
Robustness and Availability	A server-based network is robust, scalable and set-up to be always available. If a computer crashes or becomes unavailable, the network is still up. There are back up servers to take over in case the primary server fails	Most P2P networks are however not as robust and scalable. If a node crashes, the system may collapse and become dysfunctional. However, most P2P network these days are set up to ensure that the network is up and running even though a couple of peers go offline..Similarly, there is no setup for data back up
Efficiency	In a server based network, as the number of users grow, the efficiency decreases (unless more servers are added to the existing set up)	P2P network sees an increased efficiency as the number of users connected to the network increase

Thus from the comparison above, we defined some Cons and Pros of Peer-to-Peer Network Model as in the Table 2.2 below.

Table 2.2. Advantages and Disadvantages of Peer-to-Peer Network Model

Advantages	Disadvantages
+ Easy to set up and maintain, adding new peers is easy as configuration on a central server is not required + Reduces cost by eliminating the need of setting up and managing server + P2P networks are available even if one or more peers crash out. To bring down a P2P network, every peer in the network needs to be closed down. + Faster file sharing as a download request is served by multiple peers at the same time. The larger the network, the faster is the download.	– Absence of centralized server makes it difficult to back up and archive data, data is located in different machines – Weaker security features as every machine is in charge of securing itself from external threats – P2P networks make decentralization a reality. However P2P networks can become a security scare at times if one or more peers turn hostile. Most networks take security seriously these days and so the security is maintained, as in the case of blockchain.

2.2.4. Ethereum Platform

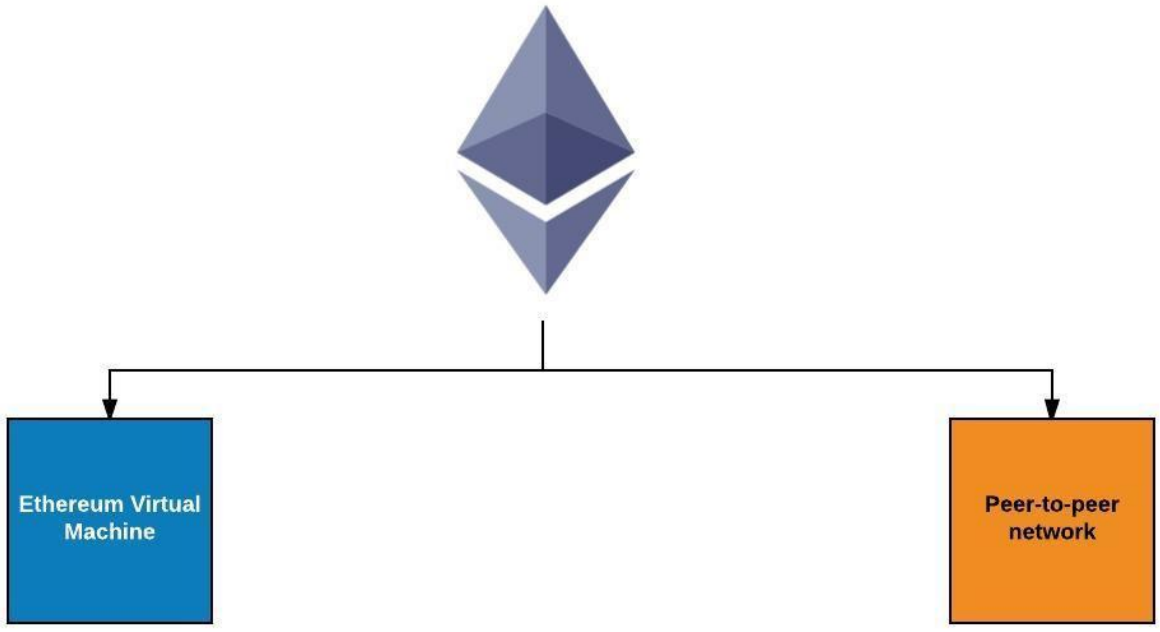

Figure 2.21. Ethereum Platform [18]

2.2.4.1. History

A new generation of Blockchain with aspirations to introduce a general, completely trustless smart contract platform called Ethereum, which is a mixture of Blockchain, Cryptography and Decentralized Contract technologies.

In 2014, the founders of Ethereum were Vitalik Buterin, Gavin Wood and Jeffrey Wilcke and includes an Ethereum Virtual Machine and a peer-to-peer network protocol. Numerous nodes connected to the Network manage and upgrade the Ethereum Blockchain database. The Ethereum Virtual Machine runs each and every node of the networks and executes the same instructions using Ethereum protocol. Ethereum could therefore be defined as a "World Machine".

2.2.4.2. Introduction to Ethereum

Ethereum is a programmable Blockchain, as has been reported. Instead of supplying users with a series of predefined operations (as Bitcoin does with its transactions), Ethereum provides them with a friendly language running on a "virtual machine"-called an Ethereum Virtual Machine (EVM). Ethereum Virtual Machine (EVM) can execute arbitrary algorithmic complexity code at the heart of Ethereum, which makes it "Turing complete"-a program or machine that recognizes a set of instructions and can execute them in a logical order, just like the computer. Using Solidity as its programming languages, Ethereum acts as a forum for various types of Blockchain decentralized applications including but not limited to.

The core network consists of nodes (or computers) linked via P2P protocol in a decentralized, peer-to-peer network. The node runs an Ethereum Virtual Machine (EVM) and executes the same instructions to ensure consensus on any particular transaction is achieved across the entire network.

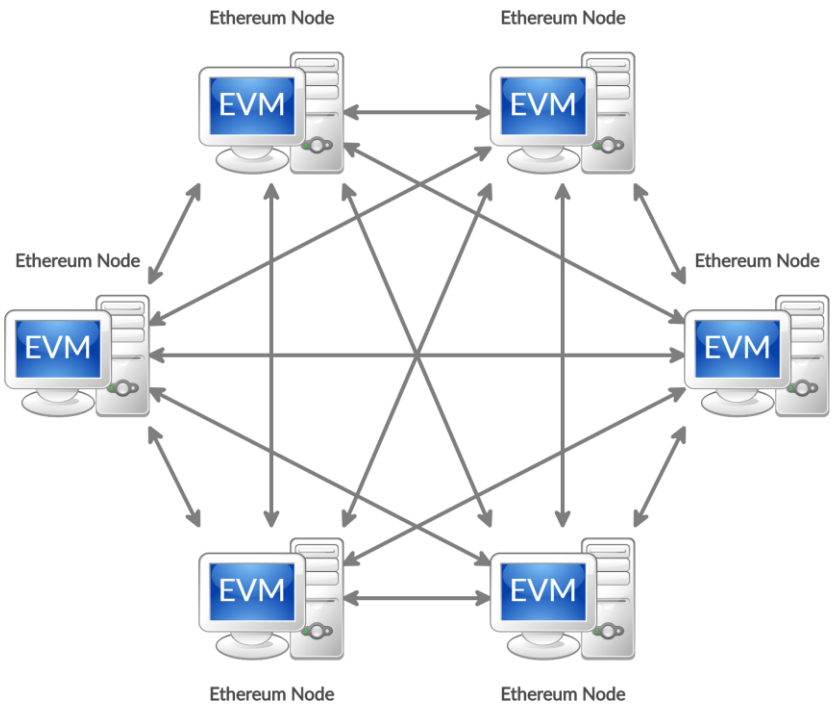

Figure 2.22. Peer-to-peer Network model with EVMs

2.2.4.3. Ethereum Virtual Machine (EVM)

The Ethereum Virtual Machine (EVM) [5] is an efficient, sandboxed virtual stack embedded into each complete Ethereum node, responsible for executing bytecode contracts. Contracts are typically written in languages of higher level, such as Solidity, and then compiled into EVM bytecode.

This means the machine code is totally isolated from the host computer's network, program file or any processes. That node in the Ethereum network runs an instance of an EVM that enables them to agree to execute the same instructions. The EVM is Turing complete, referring to a machine that can perform any logical step of a computational process. JavaScript, the programming language that power the worldwide network, makes widespread use of completeness in Turing.

Ethereum Virtual Machines have been implemented successfully in various programming languages including Go, Rust, C++, Java, JavaScript, Python, Ruby, and many others.

The 2 most popular implementation of EVM are:

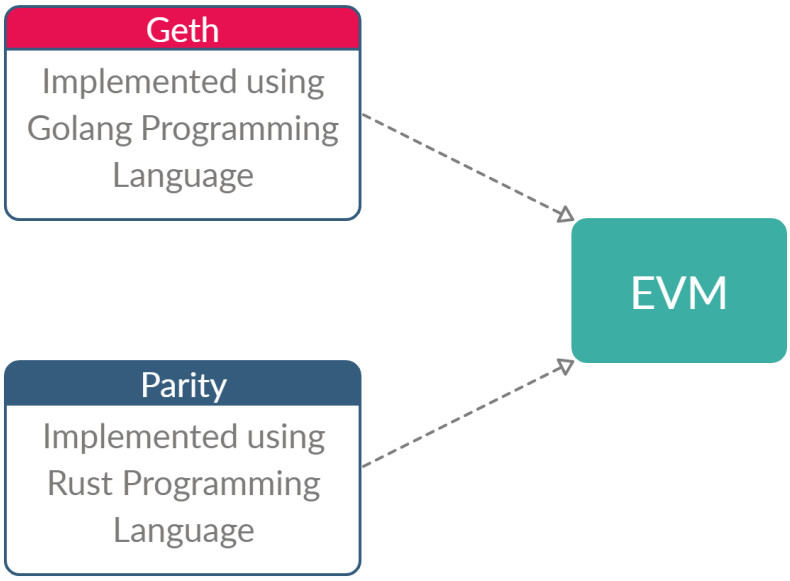

Figure 2.23. EVM's Implementation

- Geth (Go Ethereum): Geth is the main device of the Ethereum Foundation's clients. It is written in the language of programming the Go. This software contains a number of components that are worth understanding:
  - Client Daemon: It connects to other clients (also called nodes) in the network when you launch this client daemon, and downloads a copy of the blockchain. It will interact continuously with other nodes to keep up to date its copy of the blockchain. It also has the power to mine blocks and link transactions to the database, verify the block transactions and conduct the transactions as well. It also serves as a server by revealing APIs with which you can interact via RPC.
  - Geth console: This is a command line tool that lets you connect to your running node and perform various actions such as building and managing accounts, querying the blockchain, signing and submitting blockchain transactions and so on..
  - Mist Browser: This is a software for connecting with your server. Anything you can do using the geth console can be done via this Graphical User Interface.
- Parity: Parity-Ethereum is another successful implementation of the Ethereum protocol, written in the sophisticated and cutting edge language of Rust programming. It is an unofficial company and is maintained by Parity Inc. Anyone can use the client software

and become part of the Ethereum network. Parity-Ethereum's aim is to be the fastest, lightest, and most secure Ethereum client.

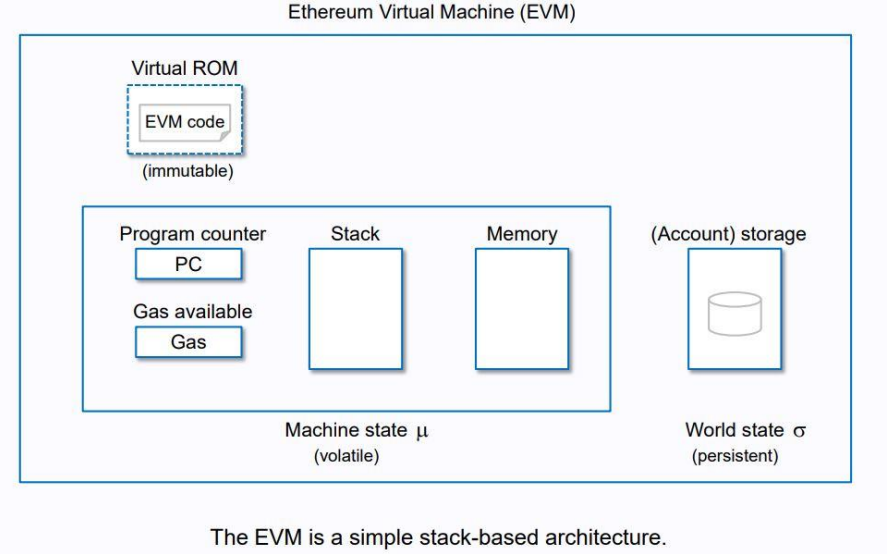

Figure 2.24. EVM's Simple Stack-Based Architecture [19]

The EVM is important for the Ethereum Protocol and is instrumental to the Ethereum system's consensus engine. This allows anyone to execute code in a trustless environment in which the result of an execution can be assured and the execution of smart contracts is entirely deterministic.

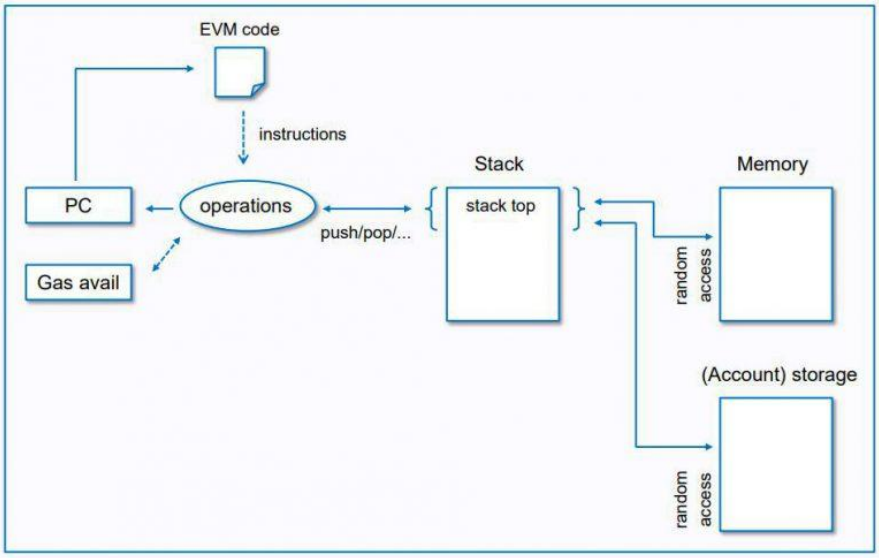

Figure 2.25. EVM's Execution Model [19]

For each instruction implemented on the EVM [6], a program that keeps track of the cost of execution assigns an associated cost in Gas units to the instruction. When a user wants to start an execution, they deposit some Ether to pay for this gas cost

Two major problems are solved by using the Gas mechanism: A validator is guaranteed to receive the initial prepaid number, even if the execution fails. An execution can't run longer than the amount prepaid will allow. Instead of endlessly looping the execution will run until Gas runs out.

When a transaction is sent to the network, the transaction may be taken by validators, executing the corresponding code. The validator must make certain:

- All transaction information is valid.
- The sender has ample funds to pay for carrying out the transaction.
- During execution the EVM did not run into any exceptions.

The EVM achieves Turing Completeness [7] by allowing an economy that charges per executed software instruction, rather than per executed financial transaction like Bitcoin. You have a sort of payment for running programs instead of a transaction fee.

Being Turing complete means that Ethereum is, technically speaking, a peer-to-peer general purpose machine worldwide, and could even presume Internet functions as we know them. Ethereum may allow us to build file-sharing economies, peer-to-peer crowd-funding events, smart contracts, markets to rent out your laptop's unused hard-drive space, a dis-intermediated Uber or Facebook (products without companies), etc. It's essentially like the 1994 internet: nobody knows what the future will bring.

2.2.4.4. Ethereum Nodes

A node is simply a computer which is part of the Ethereum network. This participation can take three forms:

- By keeping a shallow-copy of the blockchain aka a Light Client.
- By keeping a full-copy of the blockchain aka a Full Node.
- By verifying the transactions aka Mining.

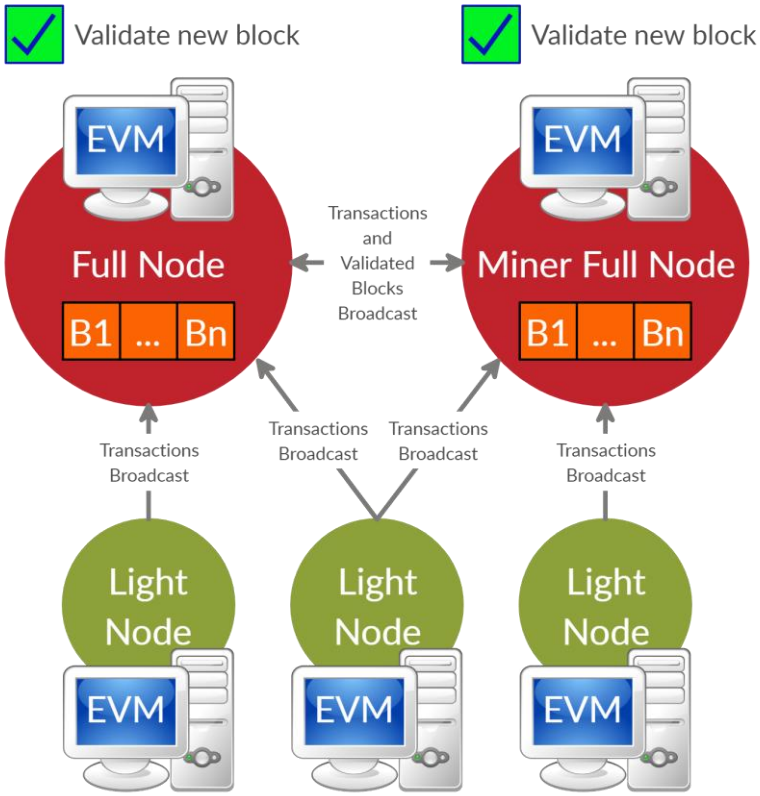

Figure 2.26. EVM Node's Broadcasting Work Flow

❖ Full Node

Any device linked to the Ethereum network, completely implementing all Ethereum's consensus laws, is considered a Full Node. A full node in the user's machine installs the entire blockchain. Full nodes form the Ethereum system's backbone, and keep the whole network truthful. Some of the consensus rules which enforce whole nodes are:

- Ensure that the appropriate block reward is issued for each block mined (5 ETH).
- Transactions have the correct signatures.
- Transactions and blocks are in the correct data format.
- No double spending is occurring in any of the blocks.
- Full nodes essentially verify nodes and transactions and relay information to other nodes (using the gossip protocol).

All miners are full nodes to keep it simple but not all full nodes are miners. To access the blockchain, miners must be running full nodes. Anyone running a full node doesn't need to check for blocks.

❖ **Light Node (Light-Client)**

As we have already described, the concept of a peer-to-peer program is to share network responsibilities between nodes called peers. Any one of them is given no preference. But what about people who want to take part in the network but don't have the system resources in their system to access and manage the entire blockchain ? We can choose to become Clients of Light. By being a Light User, they get high-security guarantees about certain Ethereum states and also the ability to check a transaction's execution.

❖ **Interacting with EVMs Node**

Interacting with Ethereum Virtual Machine requires RPCs (Remote Procedure Calls) from Client using Web3 library (which was implemented using various programming languages, such as Web3JS using JavaScript, Web3Py using Python, etc.)

On the other hand, for miners and developing purposes, interacting with EVMs using Personal Console is more preferable. Since the Console commands the Virtual Machine (or Daemon) directly without using RPCs, which is more secure.

2.2.4.5. Ethereum Networks

Each nodes in Ethereum Network communicate to each other using Ethereum protocol, the 2 most popular EVM implemented are Geth and Parity. Please review part 2.2.4.3 - Ethereum Virtual Machine for more details of the Mist / Parity Browser and Geth / Parity Console.

There are many network running based on the Ethereum Platform, included both private and public network. The table below shows all the public network including Ethereum Main-net (Real network) and Ethereum test network.

Table 2.3. Ethereum's Public Networks

Network	Network ID	Note / Consensus Mechanism
Main	1	
Morden	2	Deprecated

Ropsten	3	Proof-of-Work
Rinkeby	4	Proof-of-Authority, supported by Geth only
Kovan	42	Proof-of-Authority, supported by Parity only

2.2.4.6. Ethereum' Currency System

Ether is the standard crypto currency of the Ethereum network which can be used in trading with other supported crypto currencies. In the other hand, it can be seen as the fuel, which is needed to run a transaction on Ethereum Blockchain. Ether is also a reward to miners each time they succeed mining a new block on Ethereum Blockchain. Developers who plan to use Ethereum as a framework for creating apps need Ether, as it is used to pay computational gas fees within the EVM. Users wishing to communicate with Ethereum Blockchain applications will need to use Ether early.

Ether is to the context of the Ethereum network, just as the Bitcoin Blockchain network is to it. Therefore, users should become miners of Ethereum, or exchange with other currencies using centralized or trustless networks to obtain some Ethers. The base unit is called Wei, or the smallest unit of Ether.

$$1\ (\text{Ether}) = 10^{18}\ (\text{Wei})$$

Figure 2.27. Conversion from Ether to Wei Unit

The Figure 2.28 below shows the conversion of Ethereum' units into Wei and Ether.

Unit	Wei	Ether
Wei	1wei	10^{-18}ETH
Babbage	1,000wei	10^{-15}ETH
Lovelace	1,000,000wei	10^{-12}ETH
Shannon	10^{9}wei	10^{-9}ETH
Szabo	10^{12}wei	0.000001ETH
Finny	10^{15}wei	0.001ETH
Ether	10^{18}Wei	1ETH

Figure 2.28. Unit Conversion in Ethereum Blockchain [20]

2.2.4.7. Ethereum Accounts

Comparing to Bitcoin, one of the first and biggest Blockchain application which is purely a list of transactions, Ethereum' basic unit is the accounts. Hence, there are two type of accounts:

Table 2.4. Externally Owned Accounts & Contract Accounts

Externally Owned Accounts (EOAs)	Contract Accounts
❖ Controlled by an External party or person. ❖ Accessed through private keys. ❖ Contain Ether Balance. ❖ Can send transactions as well as 'trigger' contract accounts.	❖ Have code that executes when being triggered. ❖ Also contain Ether Balance. ❖ Can trigger other contract accounts. ❖ Live on the Ethereum Blockchain.

If a user wants to join the Ethereum community, they should have built a user account and handled the keys for this account to be run. Within Ethereum, these kinds of human-managed accounts are called Externally Owned Accounts (EOAs). Once an Externally Owned Account (EOA) has been developed, transfers can be made from that account to other account types as well as to other account types–Contract Accounts that are stationary on the Ethereum Network.

A contract account includes code for executing such planned functions, and this code is placed on the blockchain of Ethereum. When an EOA or another contract account activates a smart contract, the EVM automatically executes the code for each participating node.

2.2.4.8. MetaMask Extension

MetaMask [8] is a browser extension that lets you run DApps as an Ethereum node without being part of the Ethereum network. By default, it lets you connect to an Ethereum Node managed by Metamask Incorporation through RPCs and run smart contracts on that Node. Or else, developers could set Infura Node or their Personal Node as Http Provider [9] instead of MetaMask Node. Please review part 2.2.4.4 - Ethereum Nodes and 2.2.4.5 - Ethereum Networks for more details of the connections provided by MetaMask.

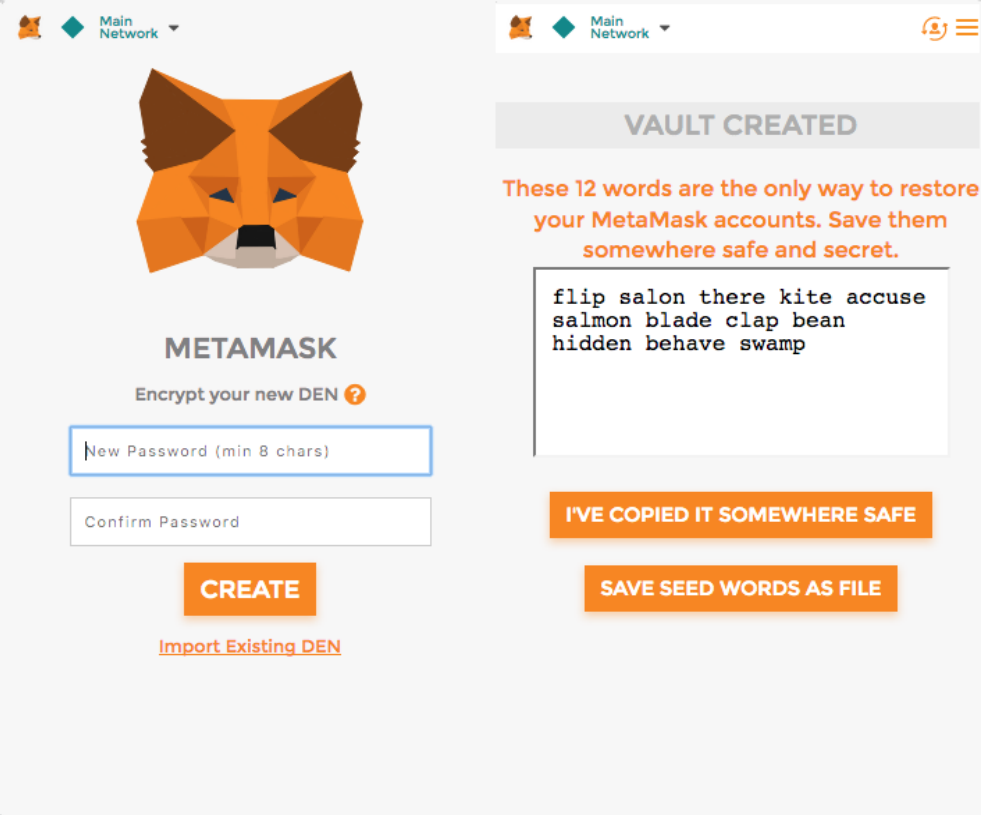

Figure 2.29. Creating MetaMask Wallet

Users can create a new Ethereum account with the DApps to send and receive Ether. Upon creating an account, you get a list of 12 terms that can be used when you forget your password to get back your account. Make sure that this is saved somewhere safe where nobody can see.

MetaMask manages your Ethereum wallet, which contains your Ethers (or money) and enables you to send and receive Ethers via an interest DApp. It is a very secure device.

Web3 object and convenience Web3.js library are inserted into the JavaScript context through MetaMask. This helps users to use their private key to manage authorisation. Whenever users make a transaction requiring a private key to sign the transaction, MetaMask will prompt the user for permission and forward the signed request to the blockchain.

2.2.4.9. Introduction to Solidity Smart Contracts

The born of Smart Contracts is one of the recent developments which are very popular nowadays. The term "smart contract" was first used by Nick Szabo in 1997 since he wanted to use a distributed ledger to store contracts.

Nowadays, smart contracts are much similar to contracts in the real world. The only difference is that they are completely digitalized. Numerous participants can agree on a set of rules, create and code a smart contract. As soon as the criterion/rules are met, it automatically authorizes the validation between two parties. Smart contracts do not store conditions only but also the data itself. However, people should accept the limitations of smart contracts because computers cannot handle everything, thus requires human supports. In the real world, a smart contract is not easy implemented since there are various of conditions to be met. Moreover, smart contracts once published can neither be altered nor changed, just a silly mistakes can be costly for both developers and users.

- There are several steps to create a smart contract in Ethereum:
    1. A smart contract is coded with Ethereum programming language (such as Solidity or Vyper) following some conditions (e.g. "If A is true, then do B") and then deployed on Ethereum Blockchain by an EOA (using some Wei as fees).
    2. Once the contract is deployed, the smart contract identify itself as a public key address, in which developers could use to reach the contract destination and trigger its code executing in every EVMs participating the network. This address is the contract account of that smart contract.
    3. A deployed smart contract can neither be changed nor tampered, even by the EOA that created it.

4. The EOA which deployed a smart contract becomes the owner of that Contract Account.

The fact is that Smart Contracts are actually a bunch of codes, which are deployed to the Ethereum Blockchain. Moreover, this code will runs on every single node connected to the Ethereum network when a transaction has been mined successfully.

❖ Each smart contract contains:

1. The Smart Contract's address, which is actually a Smart Contract Account.
2. Byte code- This is the code that the EVM executes.
3. ABI — The Application Binary Interface is the standard way to interact with contracts in every Ethereum Virtual Machine or else in the Ethereum ecosystem, both from outside of the Blockchain as users sending transactions forcing the contract to be run and inside as contract-to-contract interaction.

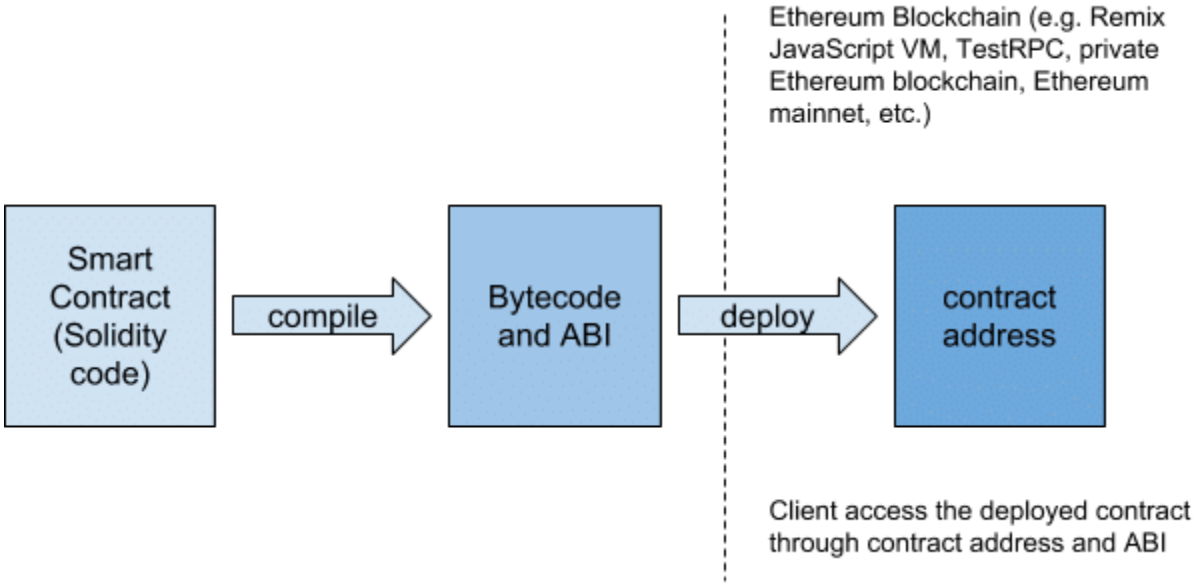

Figure 2.30. Solidity Smart Contract [21]

2.2.4.10. Solidity Programming Language

Solidity is a programming language driven and at a high level for the implementation of smart contracts. Various well-known languages such as Python, C++, and JavaScript affected solidity itself. It is however designed to run on the virtual machine Ethereum (EVM).

Solidity is statically typed, supports inheritance, libraries and complex user-defined styles.

Remix IDE (Integrated Development Environment) is currently serving as an online IDE, a web based platform at the following url: http://remix.ethereum.org/

2.2.4.11. Ethereum' Workflow

Ethereum Blockchain provides Smart Contract for developers and users to achieve consensus without intermediates. For each transaction made by users, the state of EVMs of all Ethereum Node are changed depend on the gas price the user pay. To be more specific, the more gas user pay, the faster the transaction mined. Thus changing the EVM's state and its decentralized data.

❖ **Ethereum Contract's Deployment & Interaction**

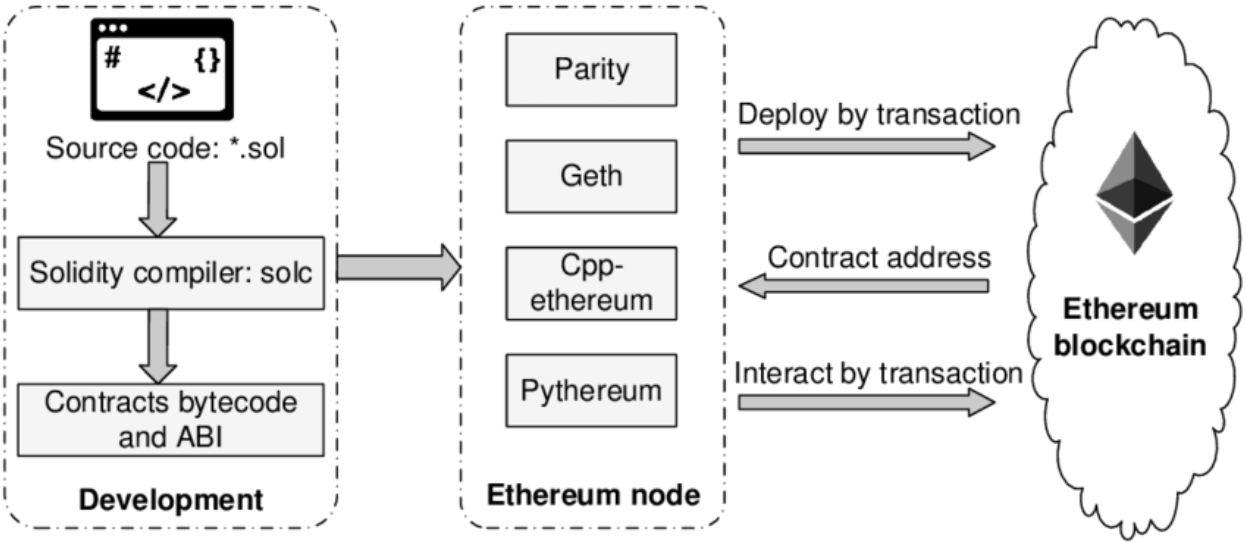

Figure 2.31. Ethereum Smart Contract Interaction [22]

❖ **Ethereum Transaction**

Ethereum has the same Hashing data and Proof-of-work mechanism as a Blockchain, but its hashing function is keccak256, sometimes (erroneously) called SHA3. On Ethereum, a user could trigger a transaction from his or her EOA.

– A transaction is a validated packet that contain:
  + **The address** of the recipient.
  + **A signature** that proves the possession of the sender's account.
  + **Value field** – The amount of value transferred from the sender to the recipient in ether unit. It can be empty, in some cases.

+ **An optional data field**, which could be a function call to some contract or an arbitrary message. As an example, accidental insurance smart contract would require to input the proportion of damage in order to release the respective compensation.
+ **Gas** is required to run transactions on Ethereum Blockchain. Essentially it's the metering system used by Ethereum that accounts for the bandwidth. In depth, on the Ethereum Blockchain Network it stands for data storage costs and computation costs. Each computing activity in the EVM uses gas and various instructions consume different amounts of gas. Those additional attributes must be generated before triggering new transactions:
    - **startgas**: the quantity of "gas" which a transaction is willing to consume in storage or bandwidth used. However, it is really difficult to define exactly how much is the startgas. Therefore, there are certain APIs support developers to estimate the amount of initial gas before deployment.
    - **gasprice**: mentioned by the transaction sender. This is the amount of ether that a sender is willing to pay per a unit of Ethereum gas. The more user pay, the faster the transactions are mined since miners will always pick your transactions first.
    - **gas_rem**: A refund to the sender who makes the transactions and will receive the left over gas if a transaction is successfully mined and consuming less gas than its specified limit, the transaction sender receives a refund of calculated formula *gas_rem*gasprice*.

Once a request is sent, either another External Owned Account or a Contract Account can be the destination. The transaction to an EOA is simply transferring Ether, meaning ether balances will be changed from both sender and receiver. However, if the transaction's destination is a Contract Account, the smart contract stored on each EVM is automatically executed. In some cases, a Contract Account needs to run a function from another Smart Contract (or Contract Account), it will send a message to that Contract Account which contains:

– The address of the contract sending the message.
– The address of recipient contract.
– The amount of ether to transfer alongside the message.
– An optional data field.
– An initial gas value for starting the execution.

The result is the contract which receives the message will automatically running its code.

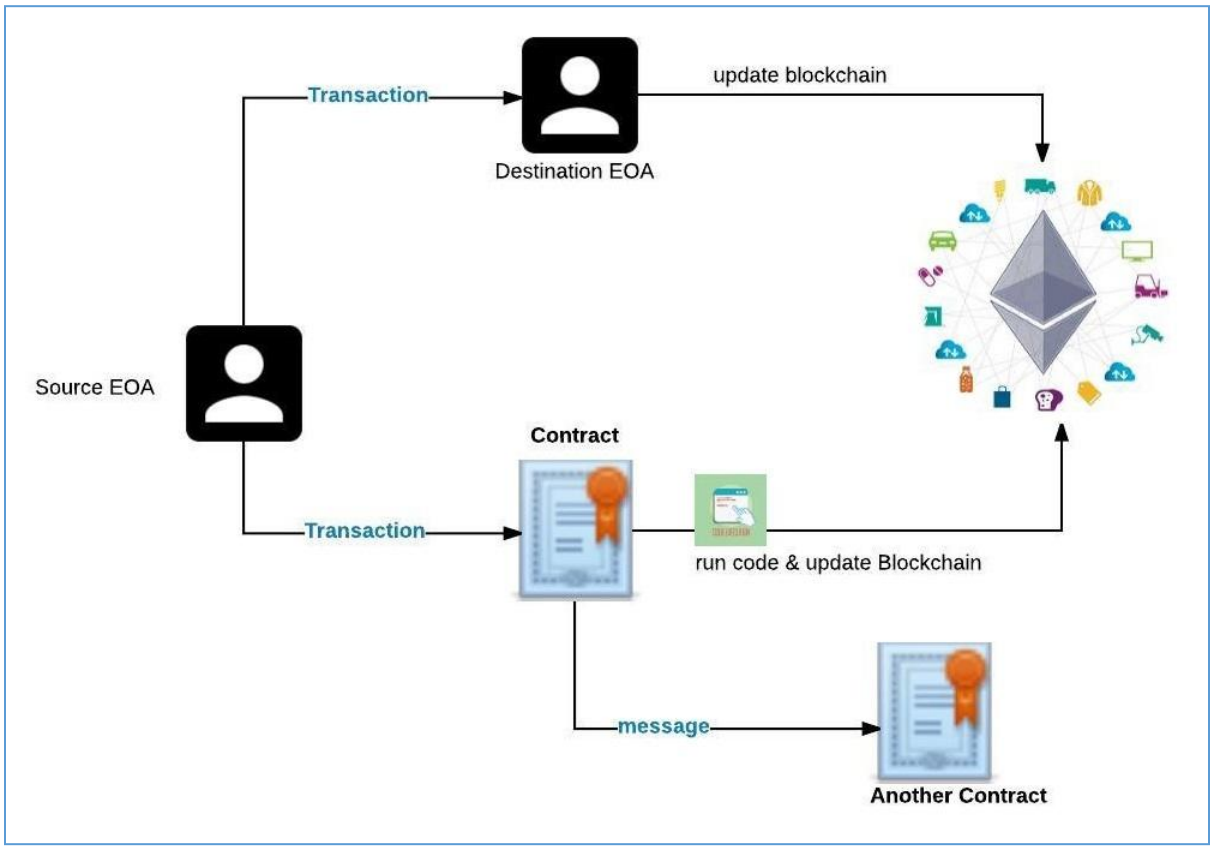

Figure 2.32. Interaction Between Accounts and Blockchain network [23]

After a transaction is done, the "state" of an account is updated to the Ethereum Blockchain. To be more specific, at the first time an account (either EOA or Contract Account) is created, it remains at the genesis state. When a transactions made by any accounts was successfully added to any block on the Ethereum Blockchain or is successfully mined by a miner (a Blockchain worker who groups transactions into block and compete with another miners for his or her block to be the next one to be added to Ethereum Blockchain), the state of the recipient accounts is then updated to the Ethereum Blockchain by the whole listened nodes on the network.

Miners are rewarded the amount of left over Ethers, which is equal to *startgas*gasprice* of a transaction. At the start of a transaction execution, this amount of ethers is immediately removed from the sender's account to make sure that miner receives the fee even if the sender account is bankrupted midway during transaction execution. If there is gas refund *gas_rem* to the sender, miner of that transaction receives a refund of *(startgas – gas_rem)* gasprice.*

2.2.4.12. Ethereum Decentralized Application (DApp)

At the core of DApps, they are software programs that take advantages of the Blockchain technology and smart contracts to achieve decentralized application. This means that any DApp cannot be controlled by any single entity since the Blockchain itself is immutable. Smart contracts are designed to enforce an agreement created between parties. Moreover, it can be used to create whole ecosystems of properties exchange within a DApp, making the use case of DApps even more extensive.

The problem DApps are trying to address is that being control over one's data. When users use the centralized application such as Google services, their actions are likely being recorded on that platform. Hence, the centralized service providers can store that data and sell it to advertisers so that marketers could tailor ads to suit users' preferences. For many people, treating their data in this manner is uncomfortable feeling. Ideally, data should remain oneself. DApps give users back control over their data, by ensuring that no single entity is in charge of any user's data.

An application that reaches fully decentralized must meet the following requirements:

- **Open Source**: The code source of application must be widely public to the scrutiny.
- **Decentralized**: The data must be cryptographically stored on a public and decentralized platform such as IPFS and Swarm Blockchain Network.
- **Incentive**: Tokens or digital assets of the application must be utilized to reward its network supporters.
- **Protocol**: A tokens or digital assets must be generated using a cryptographic consensus algorithm to indicate the proof-of-value (e.g. proof-of-work or proof-of-stake).

The Ethereum Blockchain is one the most well-developed network for facilitating the creation of DApps. Using Solidity, developers can build any DApps on just about any requirements made by users. The DApps that have been produced so far speaks volumes to where this part of the ecosystem can take us to.

❖ Components of Ethereum DApps

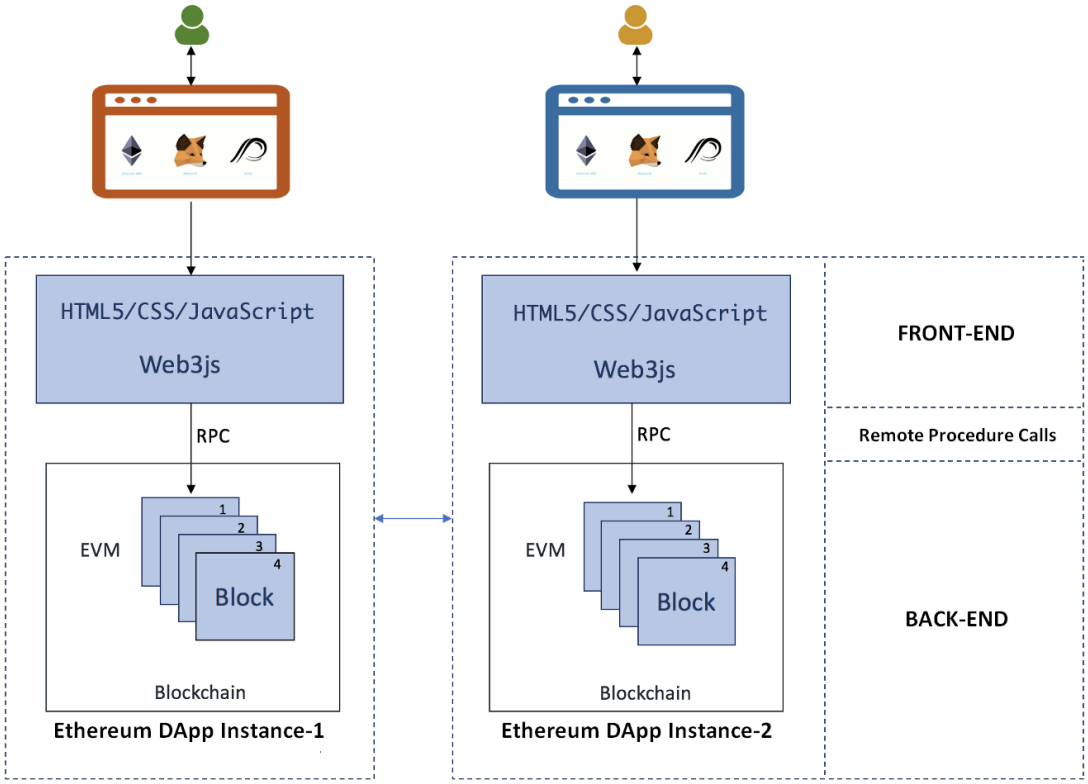

Figure 2.33. Core Components of Ethereum Decentralized Applications [24]

❖ FRONT-END / CLIENT SIDE

Front end of the Decentralized applications are typically created as web pages or desktop applications using JavaScript/HTML/CSS. It is used to handle with users inputs and display responses from the Back-end (or Ethereum Blockchain). Since there are several JavaScript frameworks, developers feel free to use any of JavaScript frameworks they want.

Web3.js is a JavaScript API that allows clients to interact with the Blockchain, including making transactions and calls to smart contracts. This API abstracts the communication with Ethereum Clients, allowing developers to focus on the content of their application. Developers must have a Web3 instance imbedded in their browser to do so.

❖ RPC

A remote procedure call (RPC) in distributed computing is the point at which a PC program triggers a procedure (subroutine) to be performed in an alternate location space (generally on another PC in a shared network), which is coded as if it were a standard (local) procedure call

without the programmer specifically coding the specifics for the remote interaction. That is, the programmers will effectively write the same code to the executing program whether the subroutine is either local or remote. This type is a client-server interaction (caller is the client and executor is the server), usually implemented via a message-passing request-response system.

Remote procedure calls (or RPC) in Ethereum platform is a connection between a local computer and Ethereum Blockchain. This connection can be archived easily by becoming a full-node of the network or just a light one. For most users, a light client along with its protocol is the best choice. The aim of the Light Client protocol is to allow users to participate in low-capacity network environments (such as embedded smart property environments, mobile phones, browser extensions, desktop application, etc.) to maintain high-security state assurance of some particular part in the state of the EVM, or to verify transaction execution. Although full security features are only available for the full nodes, the Light Client protocol also allows light nodes processing around 1 KB of data every 2 minutes to access part of the data from the network that concerns them. In addition, light customers ensure that the data provided by most miners is correctly followed. At least one honest full node is verified to be exists with the light client protocol. This means that it doesn't store all of the Blockchain data, and depends on asking the network for the data it needs every time.

❖ BACK-END / ETHEREUM BLOCKCHAIN

Ethereum Blockchain and Smart Contracts serve as the logic and storage back end. A contract is written in Solidity, a smart contract language, and is a compilation of code and data on the Ethereum Blockchain at a specific address. It's very similar to a class in Object Oriented Programming, where it includes functions and state variables. Smart Contracts, along with the Blockchain, are the basis of all Decentralized Applications. They are, like Blockchain, immutable and distributed, which means modifying them will be a stiffness if they have already deployed on the Ethereum Network. Fortunately, there are some ways to do that.

The Ethereum Virtual Machine (EVM) performs the internal state and computation of the Ethereum network as a whole. Think of the EVM as a massive decentralized computer that contains "addresses" that are capable of executing code, changing data, and interacting with each other. Thus, hard forking the whole system will eventually resulted in our desirable outcome.

Unfortunately, controlling a large proportion of nodes and convincing them hard-forked along with their satisfaction is a bit difficult.

The Figure 2.34 below illustrates an Ethereum fully decentralized application.

- The Compiled Contract's ABI formatted as JSON are stored on Swarm Decentralized Network. Clients use this ABI and the contract's address to interact with contracts in EVMs.
- The UIs, or Client Sides are served with IPFS (Inter Planetary File System).
- The HTTP Provider serving as an Ethereum Node or EVMs to execute transactions.

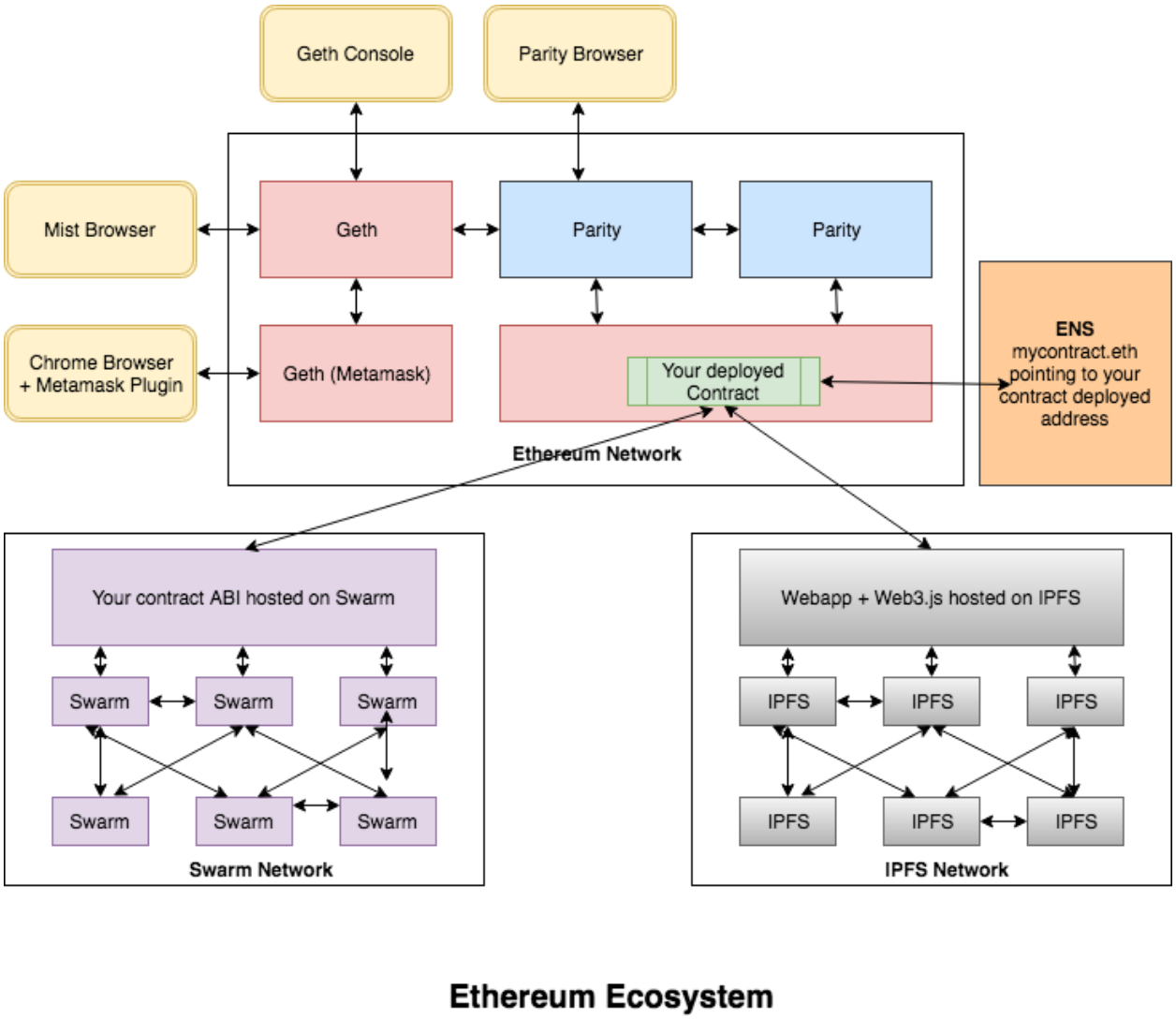

Figure 2.34. Ethereum Fully Decentralized Applications [25]

CHAPTER 3. SYSTEM REQUIREMENT

3.1. System Description

❖ Since the database is a special one, Ethereum Blockchain, the main requirement of the system must reach these criterions:

- The system must be connected to the Ethereum Blockchain test network, in this thesis paper, Ropsten..
- The system must be implemented to work along with Wallet Provider for better user's experiences, in this thesis paper, MetaMask - a Chrome Extension.
- The system must be injected with dependencies to interact with Ethereum Blockchain network, in this thesis paper, Web3JS.

❖ User Interfaces must be clear, simple, intuitive and responsive to help users finish the task with less effort and time, thus Single Page Application (SPA) is applied for Client Side in this thesis paper.

❖ The performance of Client Side must be adaptive to the drawback of SPA, thus in this thesis paper, Server-Side Rendering for Client is applied.

❖ APIs must be simple to store compiled Ethereum contracts as JSON format, thus ExpressJS is applied for Server Side in this thesis paper.

❖ User will no longer afraid of tampered data, scammers or fake commitments since the Ethereum Blockchain will tracking their properties ownership autonomously.

3.2. System Functionality

Figure 3.1. System's Entity Relationship

The functionality of the system is described as Table 3.1 below.

Table 3.1. System's Functional Description

	Function	Description
Front-End / Client Side	Searching house with different categories	Allow user to filter the houses matched requirements
	Selling house with custom input	Allow user to sell house with custom house's information
	Pending Interest	User could interest a house for owner to sign contract
	Buying house immediately	User could buy the house immediately by fully payment
	Buying house by instalment	User could buy the house partially overtime of 3 / 6 / 9 months by instalment
	Changing house's status	User could change the house's information and status, which stop house from being into new transaction
Back-End / Server Side	Compiling all Solidity Contracts	Compiling all Ethereum Contracts and exporting output as JSON format
	Deploying Admin Contract to Ethereum test net	Deploying the Administrator Contract that tracking address of all House Contract
	Response JSON Contracts to Client Side	Response compiled House, HouseAdmin contract and its address as HTTP response.
Ethereum Blockchain	Managing all Houses contract	Buying house or Selling house property is forwarded to HouseAdmin contract
		Tracking for the existence of

		all houses
	Verify House's Ownership	Always check for house's ownership when performing any transactions.
	Executing Client Side RPC request	Internally executing Client Side Requirement as Client sends transaction to Blockchain Network using its injected RPC, or Web3JS

3.3. System Workflow

The next 6 figures below illustrate sequence diagram for each of the use case, or else, each of the different buttons the Workflow above, which are [Search, Submit House, Buy Now, Pay by Installment, Rent House, Change House's Status].

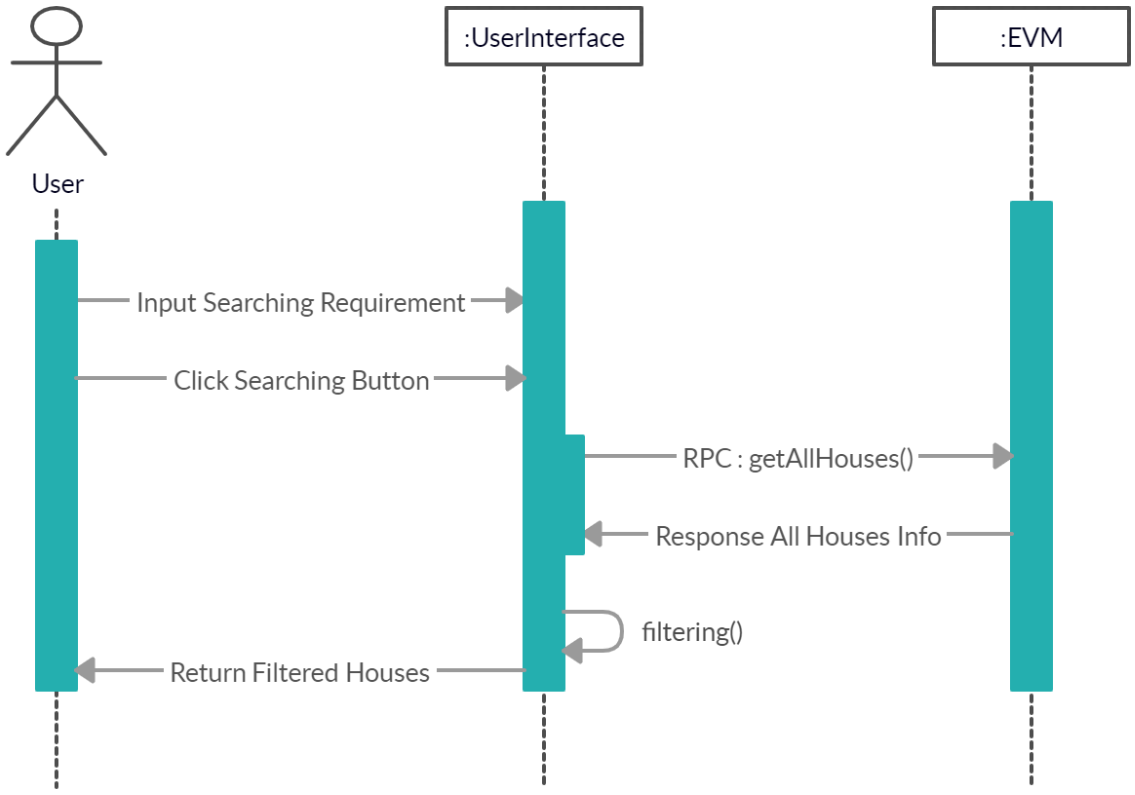

Figure 3.2. Searching Houses - Sequence Diagram

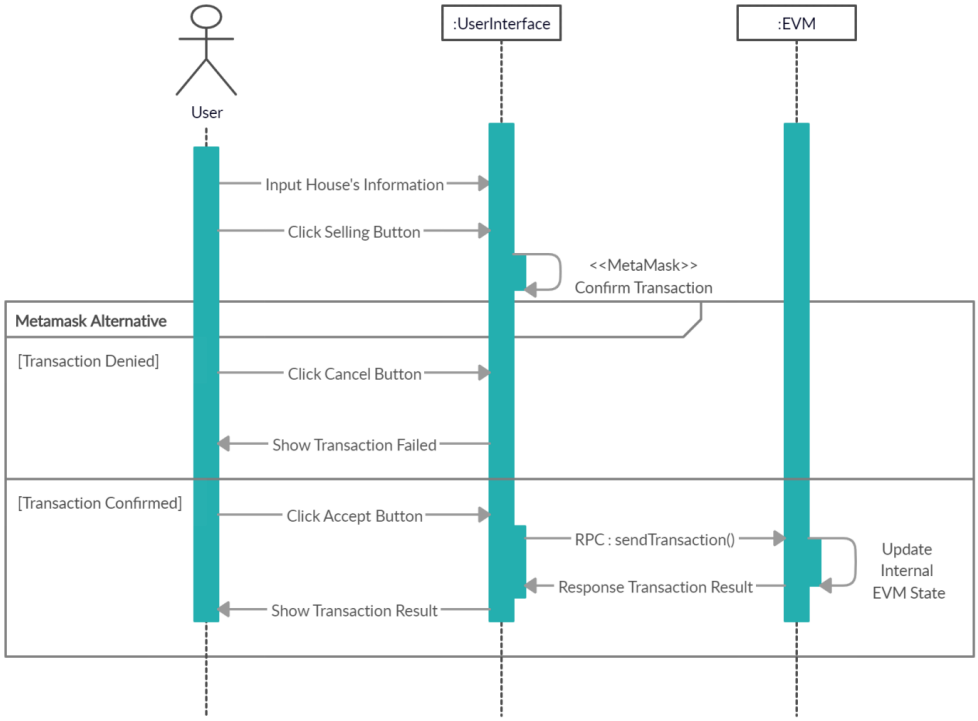

Figure 3.3. Selling Houses - Sequence Diagram

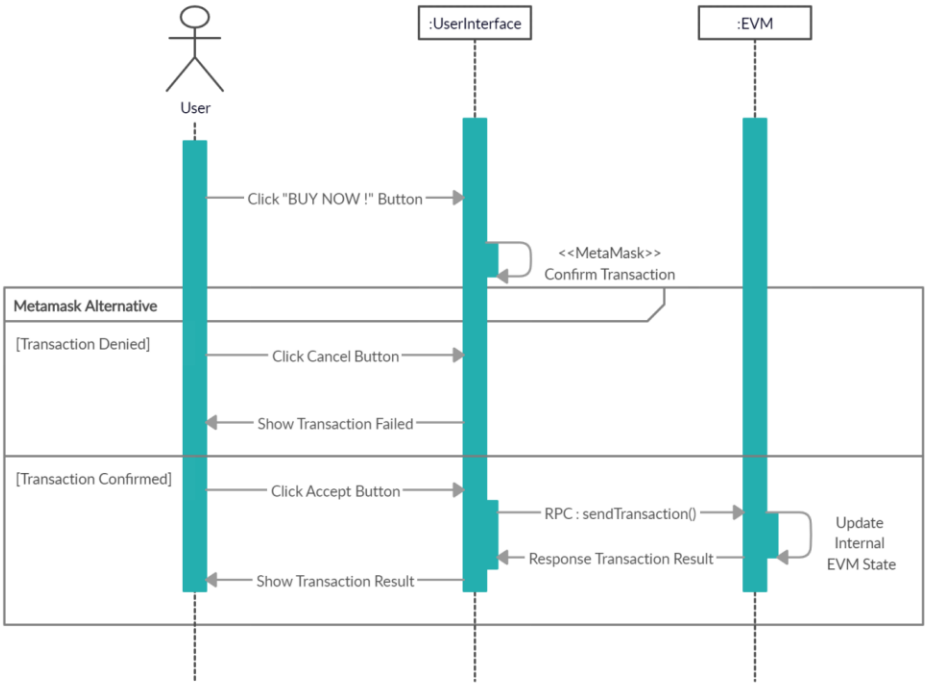

Figure 3.4. Buying Houses - Sequence Diagram

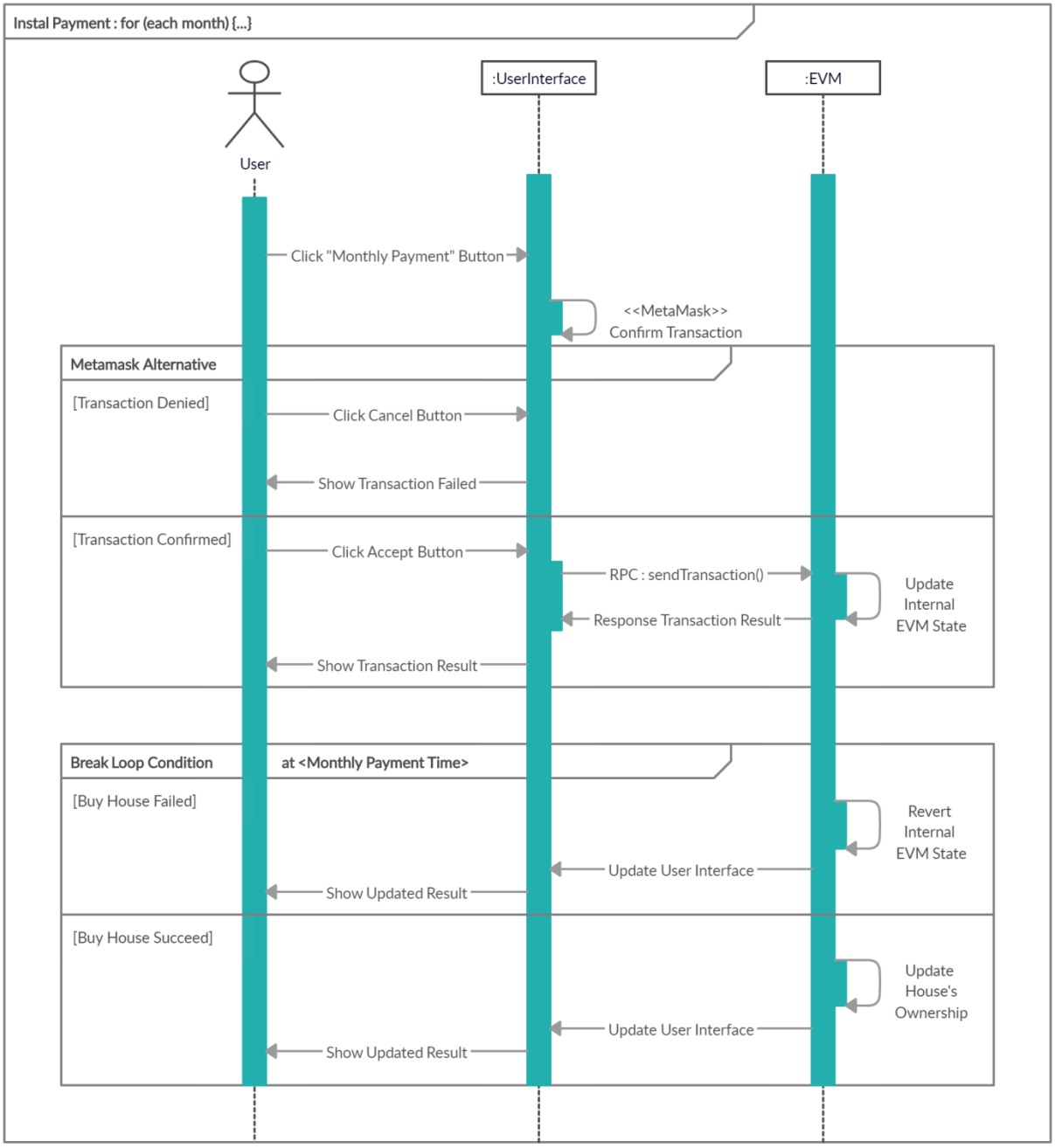

Figure 3.5. Buying Houses by Installment - Sequence Diagram

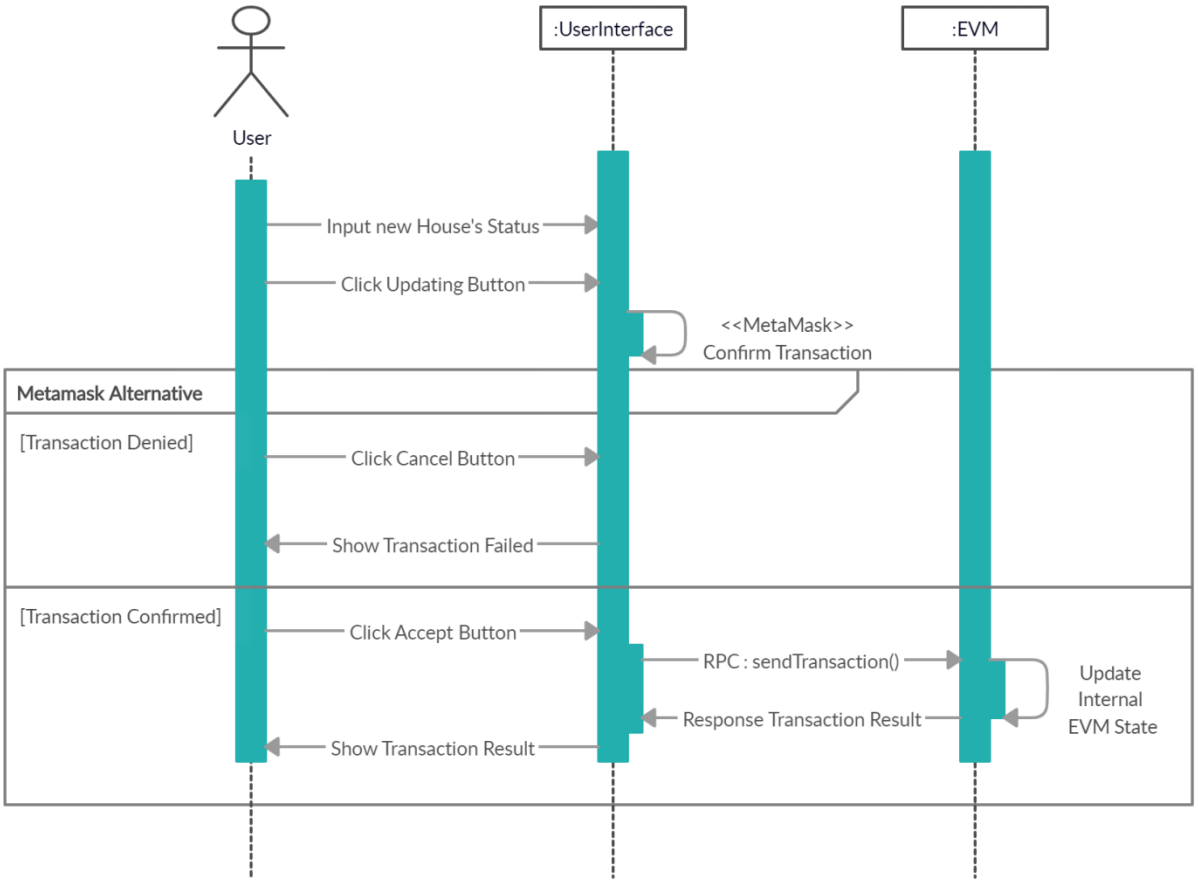

Figure 3.6. Updating House's Status - Sequence Diagram

The Figure 3.7 below illustrates the main workflow of the system.

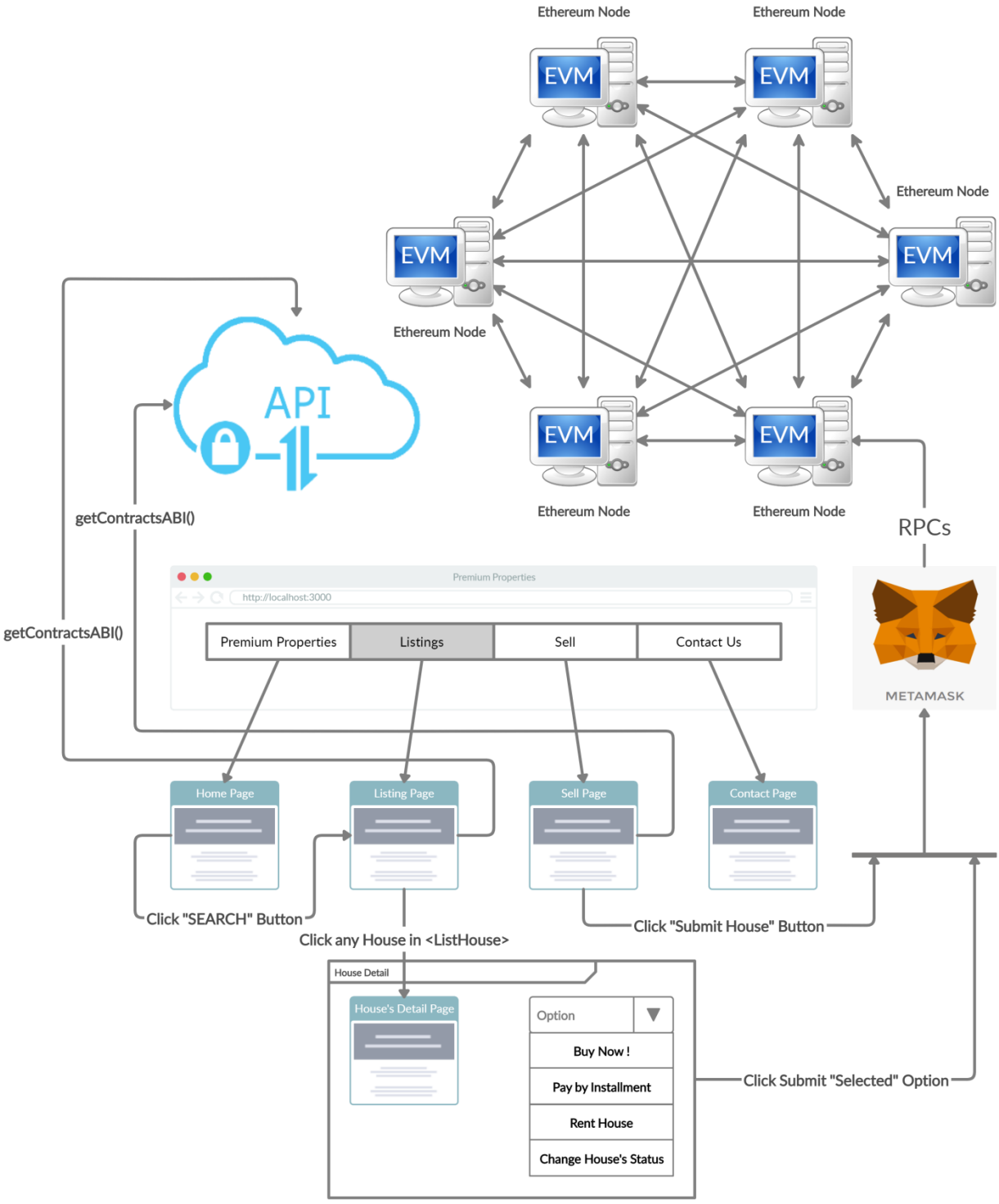

Figure 3.7. System's Workflow

CHAPTER 4. SYSTEM ARCHITECTURE

4.1. Overall Architecture

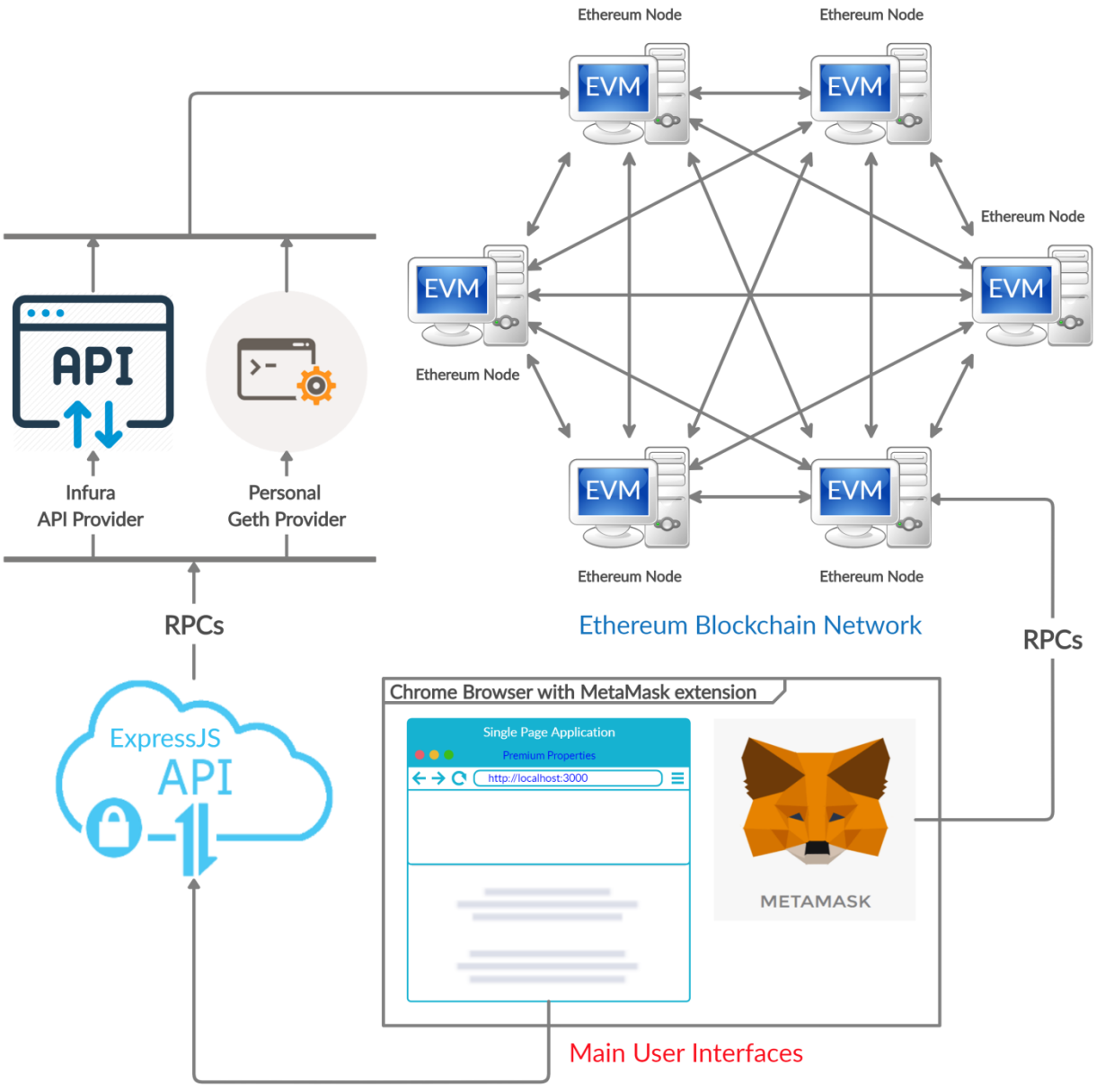

Figure 4.1. System Overall Architecture

4.2. Front-End / Client Side Architecture

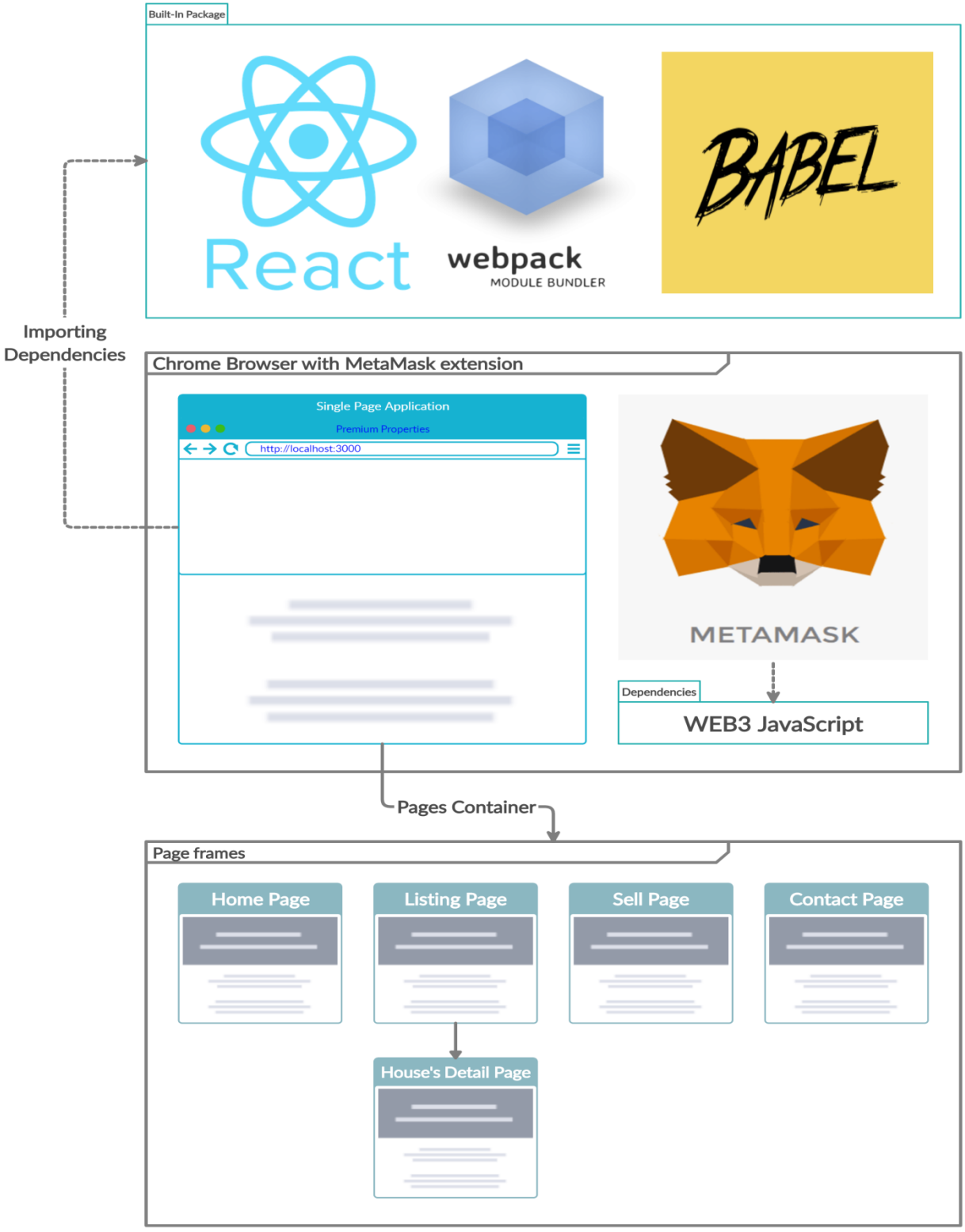

Figure 4.2. Front-End / Client Side Architecture

4.3. Back-End / Server Side Architecture

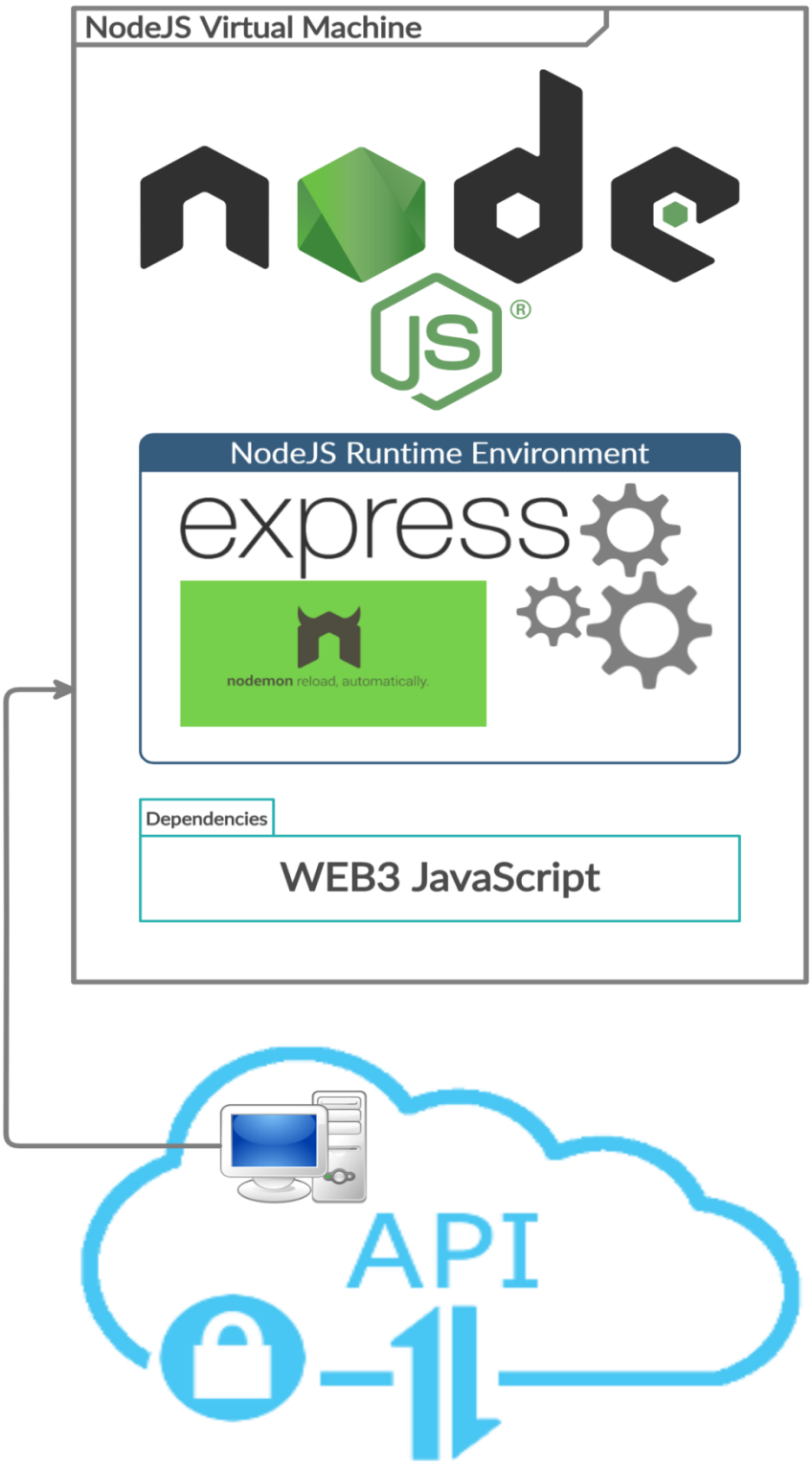

Figure 4.3. Back-End / Server Side Architecture

4.4. Blockchain / Database Architecture

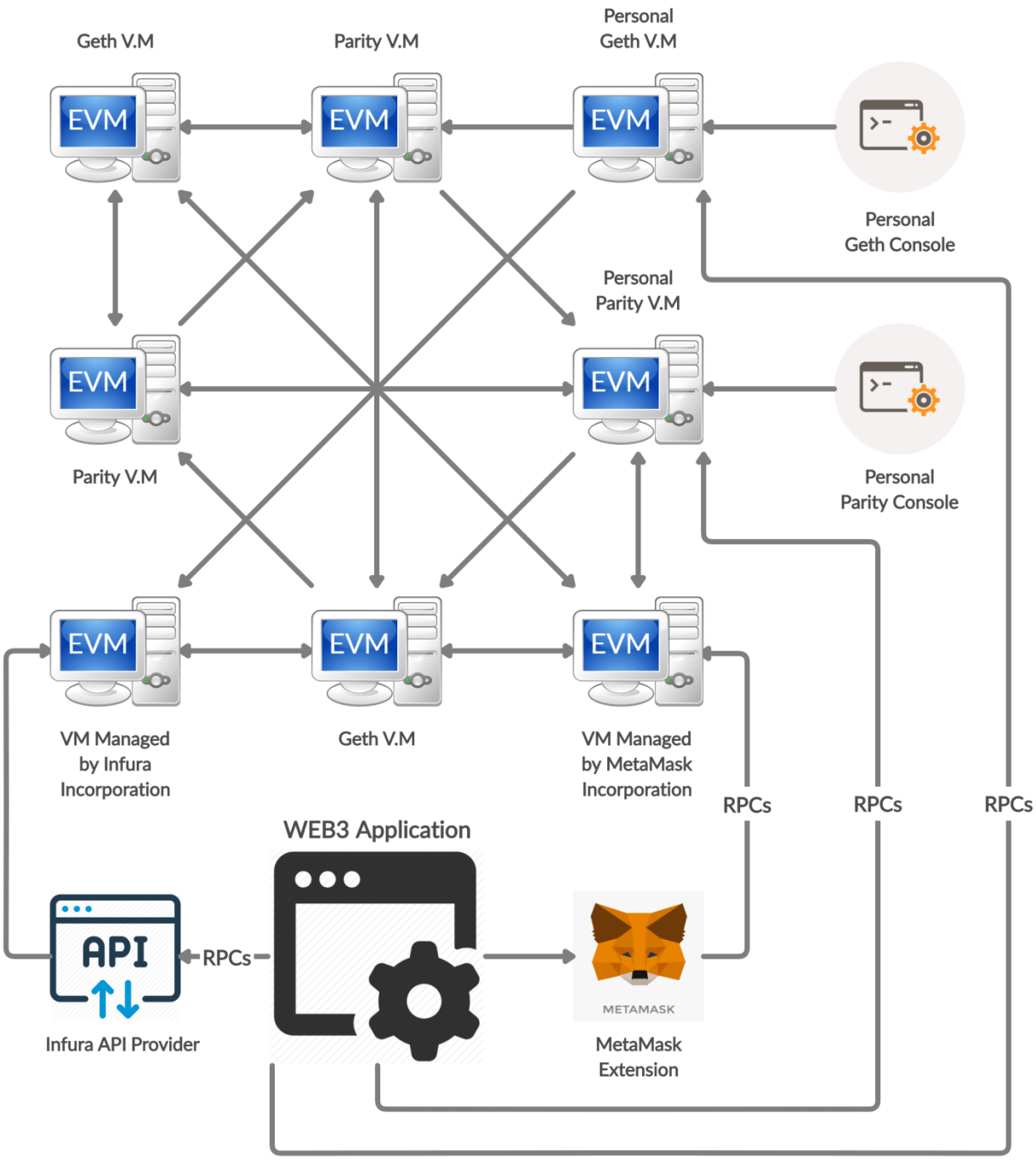

Figure 4.4. Blockchain / Database Architecture

CHAPTER 5. SYSTEM DESIGN

5.1. Front-End / Client Side

5.1.1. User Interface (UI)

The User Interfaces included 5 main pages, which is illustrates as 5 figures below.

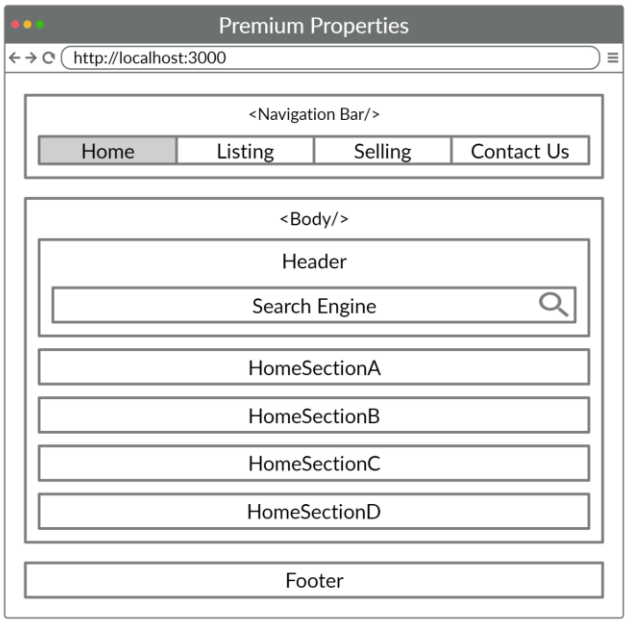

Figure 5.1. Home Page

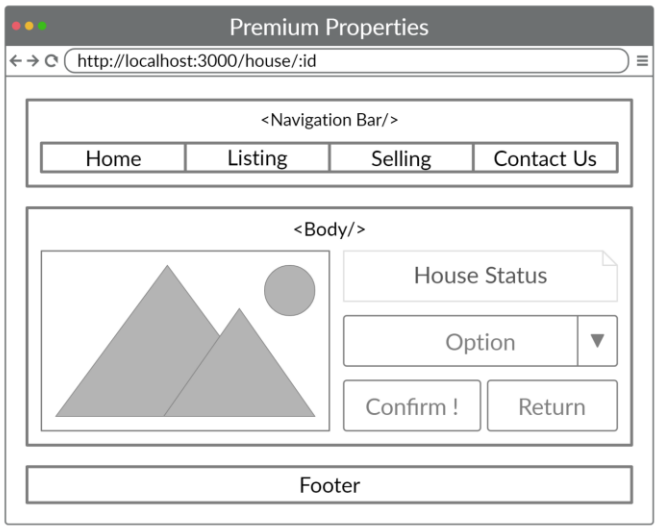

Figure 5.2. House's Detail Page

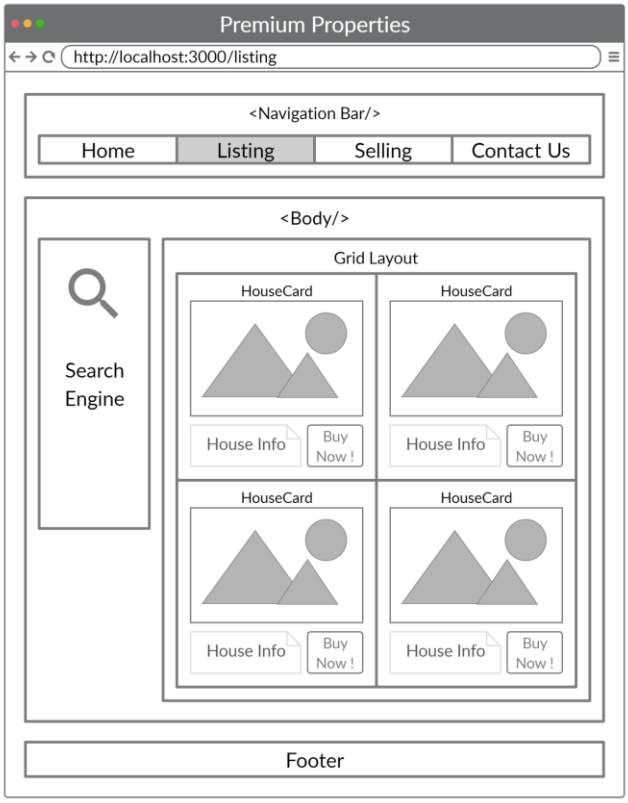

Figure 5.3. Listing Page

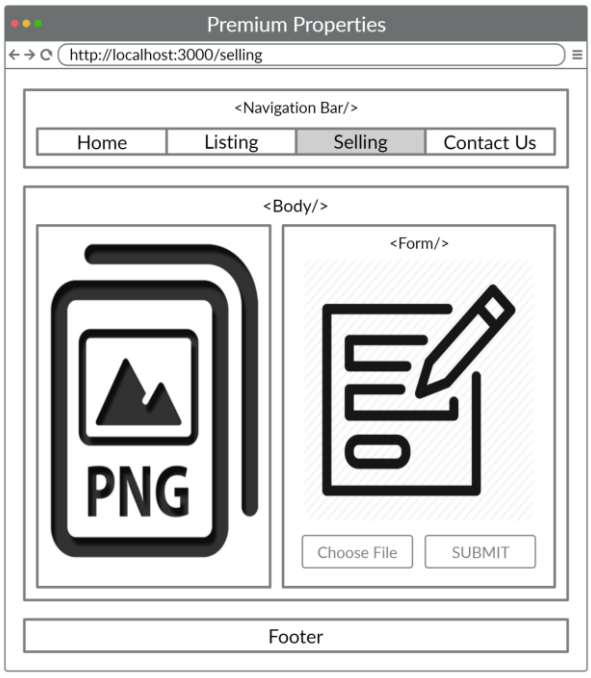

Figure 5.4. Selling Page

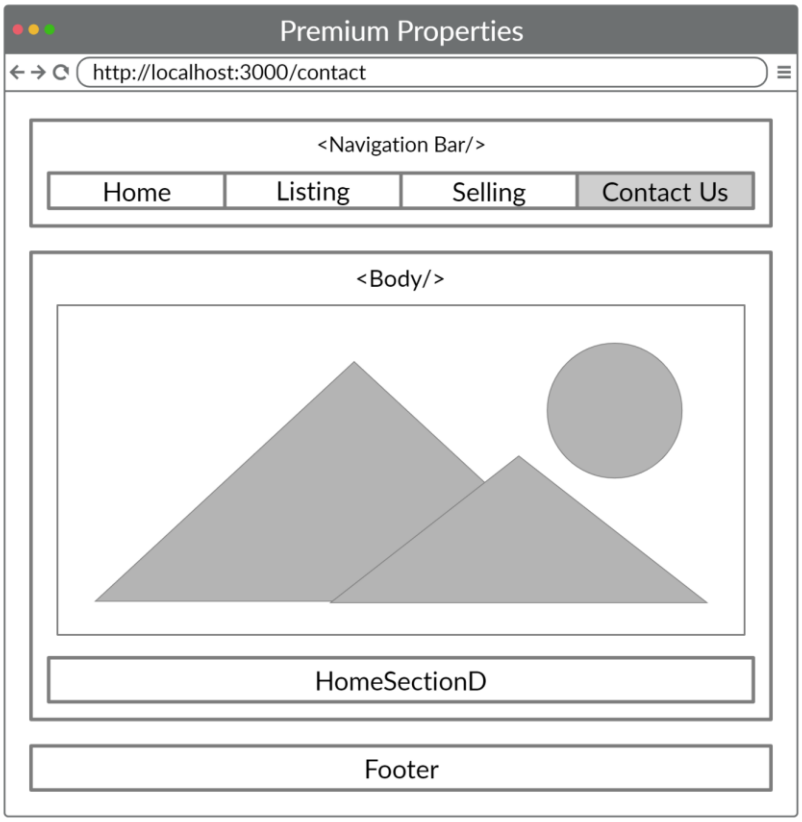

Figure 5.5. Contact Page

5.1.2. Ethereum Interaction

By installing MetaMask, the browser is automatically injected with Web3.js library and be ready to serve as a Web3 Application, hence users could easily interacting with Ethereum Blockchain Network through MetaMask.

Metamask will use Web3 object, which was injected to running environment as the browser started, to interact with the Ethereum Blockchain Network.

Please review part 2.2.4.11 - Ethereum' Workflow to understand the main flow of Ethereum Interaction between Developers / Users and Solidity Smart Contract.

Client-Side will wrap the Contract's ABI (JSON Formatted) and Contract Address (String) into new Web3.Contract object. Hence, the Clients can now interact with deployed contract using RPCs to the remote EVMs, or else, an Ethereum Node. See **Error! Reference source not ound.** below for a fully illustration.

5.2. Back-End / Server Side

The Figure 5.6 below shows the total functionality of the designed Express API. The API mainly store, compile and deploy Solidity Contracts to serve the Client Side, or else, to serve users at the end point.

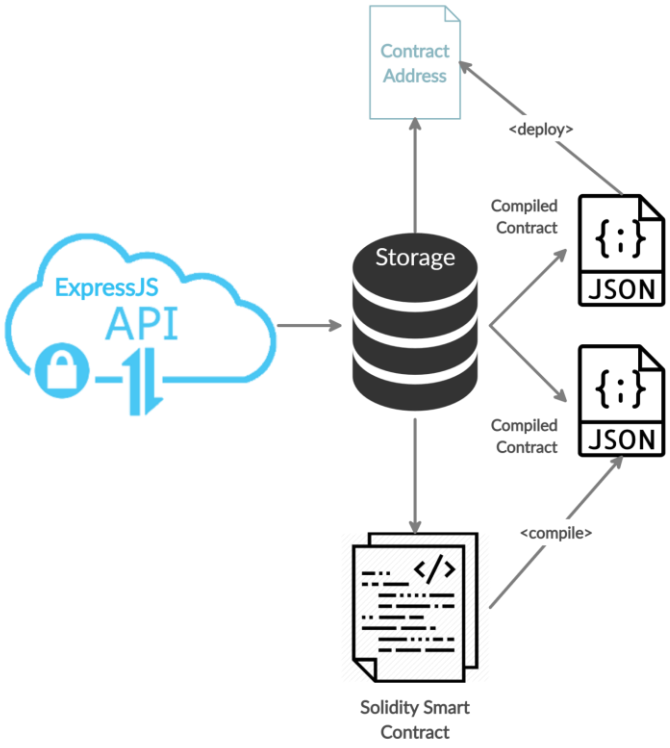

Figure 5.6. Node-Server's Main Flow

5.2.1. Application Programming Interface (API)

- The API must fulfil the System Requirement, which are:
  - + Compiling all Solidity Contracts.
  - + Deploying Admin Contract to Ethereum test net.
  - + Response JSON Contracts to Client Side.

❖ **NodeJS Server as Contract Compiler**

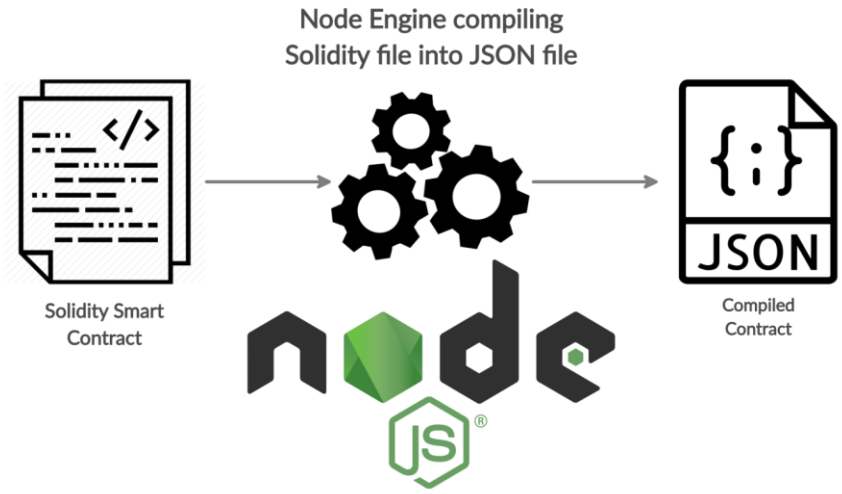

Figure 5.7. Node-Server as Compiler

❖ **NodeJS Server as JSON Contract Responder**

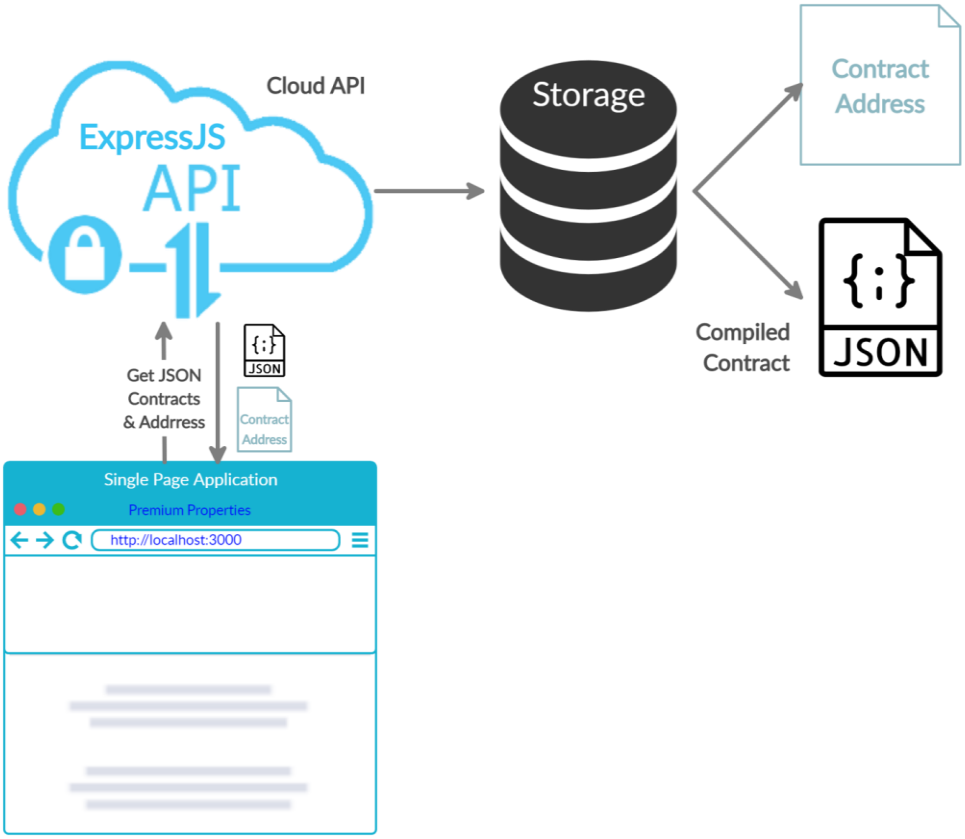

Figure 5.8. Node-Server as Storage for Client Request

❖ NodeJS Server as Contract Deployer

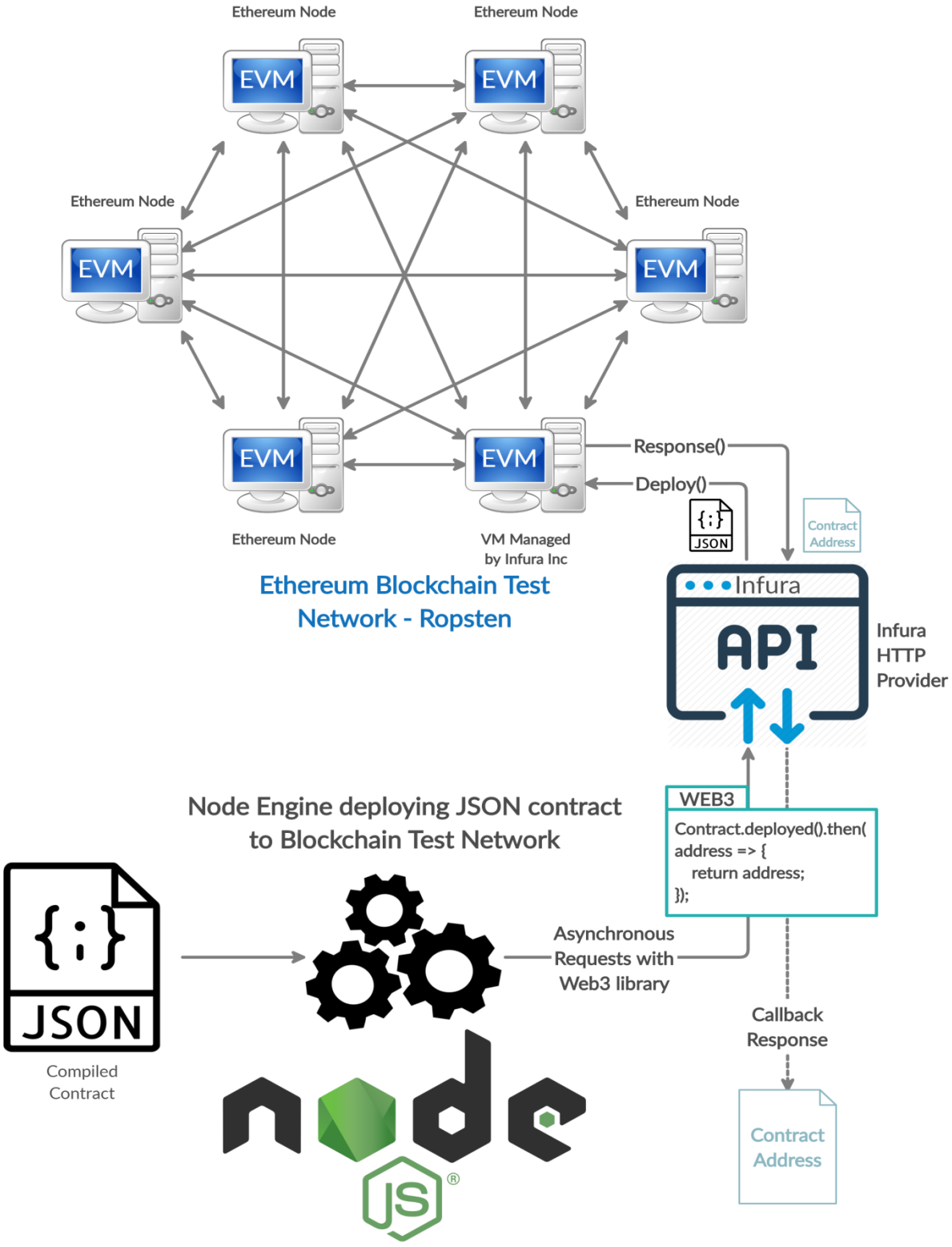

Figure 5.9. Node-Server as Deployer

5.2.2. Ethereum Interaction

The server mainly interacts with Ethereum Blockchain through Infura API Provider. Infura provides an EVM, or else, an Ethereum Node for developers to connect to the Blockchain Network.

See Figure 4.1. System Overall Architecture, Figure 4.4. Blockchain / Database Architecture and Figure 5.9. Node-Server as Deployer to understand the main flow of Server Side with Infura HTTP Provider.

5.3. Ethereum Configuration

5.3.1. Ethereum Network

The network used is described in CHAPTER 3. System Requirement, which is Ropsten Test Net, which is using Proof-of-Work as the main Consensus Mechanism for the whole Network.

You'll need a wallet provider (like truffle-hdwallet-provider) to sign your transactions before they're sent to a remote public node to deploy via Infura. Infura accounts are freely available on: https://infura.io/register

❖ **Network Setting for Server Side**

Table 5.1. Server Side customs Ropsten Network

Ropsten Test Network	Value	Description
HTTP Provider	new HDWalletProvider (String mnemonic, String infura_key)	- mnemonic: the twelve word phrase the wallet uses to generate public/private key pairs. - infura_key: HTTP Provider supported by Infura API to connect to an Ethereum Node managed by Infura.
Network ID	3	Ropsten's Id.
Gas	5500000	Ropsten has a lower block limit than main network.
Block of Confirmation	2	Number of confirmations to wait between deployments. (default: 0)

5.3.2. Ethereum Accounts

The accounts are freely generated using Hierarchical Deterministic Wallet technology [10].

The accounts used in this thesis paper are generated from the 12 random words. For example "eight one nine two eleven four three five six twelve seven ten". The amount of private keys, or accounts, can be created are up to 2 billion from just a small word phrase of 12 words.

5.3.3. Smart Contracts' Structure

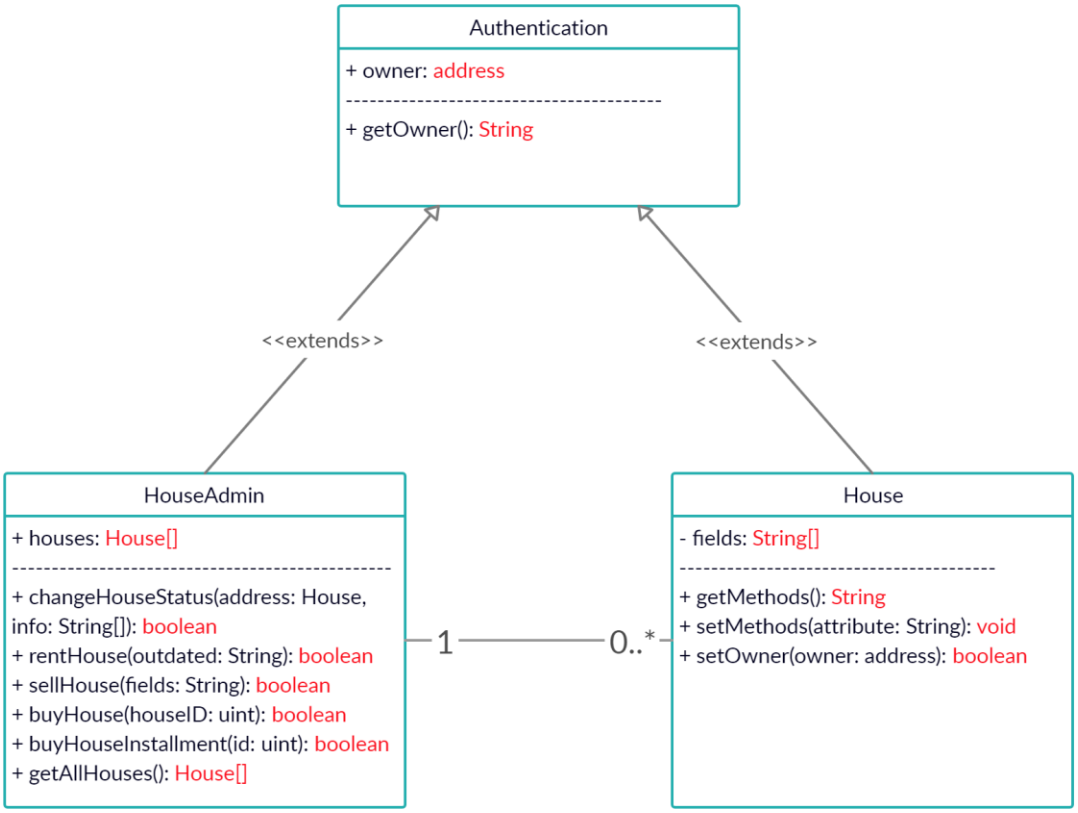

Figure 5.10. Smart Contracts' Relationship

Since Smart Contracts are written in Solidity, a high-level Programming Language. It's possible to illustrate the relationship between classes with UML notation.

Each Smart Contract could be self-executed under Ethereum Virtual Machine when a signed transaction has been forward to any Ethereum Node. Thus update the EVM's internal data in a valid way and broadcast the transactions along with changed data under consensus mechanism.

CHAPTER 6. IMPLEMENTATION

6.1. Setup Environments

6.1.1. Operating System

This implementation is based on Windows 7 Professional 64 bits.

❖ **Windows Command Line Terminal**

To open this CLI, click on Windows button, then type "cmd" inside searching box. When the program named "cmd.exe" appears, clicking on to open it normally or Run as administrator, which is your option.

After open the command prompt, change directory to your project using "cd" command. Firstly, type <D:> to move to disk D. Then cd to <Your_Project_Folder> as in **Error! Reference ource not found.** above.

- <D:>
- <cd Work/Thesis_ThinhHuynh_2019>

6.1.2. Required Tools

The Figure 6.1 below shows all contained folders inside <My_Project_Folder>, or else, <Thesis_ThinhHuynh_2019>.

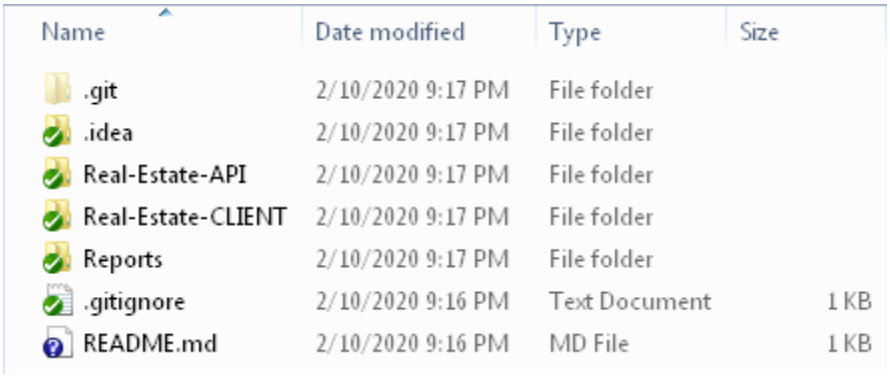

Figure 6.1. Thesis Folder

6.1.2.1. NodeJS and NPM

Node.js is a software platform that helps to create asynchronous and event-driven network applications. It contains built-in HTTP server libraries that allow developers to create their own web server and additionally build highly scalable web applications. Node.js is the first downloadable application since it offers the framework on which a Decentralized Application is built. You can download it from our website:

https://nodejs.org/en/download/

6.1.2.2. ReactJS with Babel and Webpack

Let first <cd> to <Real-Estate-CLIENT> and run.

- <npm install>

Then let's wait until all dependencies specified in package.json file are successfully downloaded into <YOUR_PROJECT_FOLDER>/Real-Estate-CLIENT/node_modules folder.

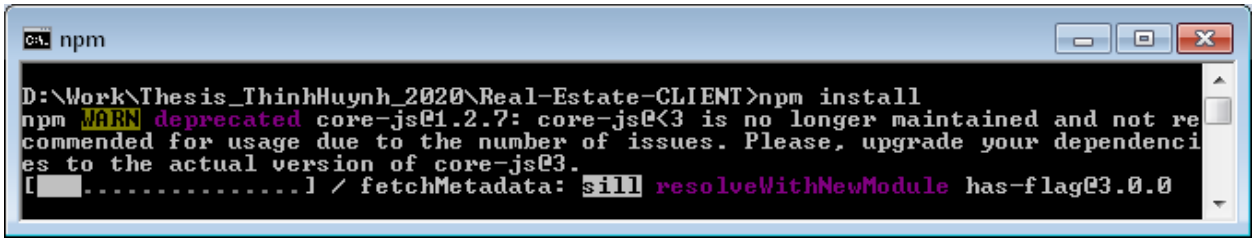

Figure 6.2. NPM Install for CLIENT

6.1.2.3. ExpessJS with Nodemon

Let first <cd> to <Real-Estate-API> and run.

- <npm install>

Then let's wait until all dependencies specified in package.json file are successfully downloaded into <YOUR_PROJECT_FOLDER>/Real-Estate-API/node_modules folder.

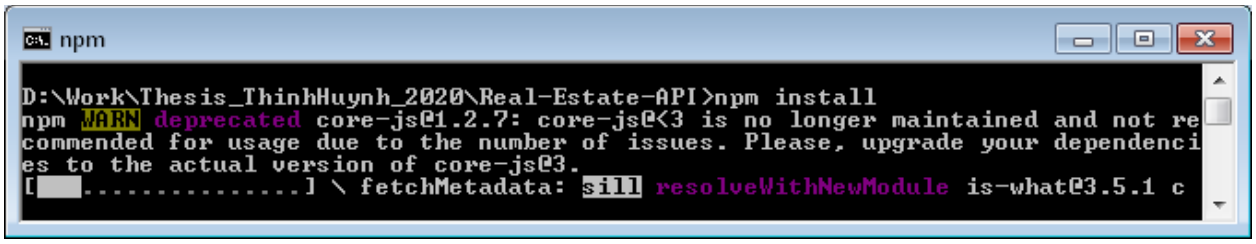

Figure 6.3. NPM Install for API

6.1.2.4. Metamask

Metamask is the standard Ethereum web3 API. It is an add-on of our web browsers such as Firefox or Chrome. There for, installing Metamask is quite simple. With Metamask, every user can interact with Ethereum Blockchain (Main net, test net or even private Blockchain) without becoming a full node. A light node user is not required to install Geth or Ganache to use Ethereum DApps. To playing with Ethereum DApps, users now only need to install Metamask add-on on their web browsers, custom the RPC URL of it and register account.

- Step 1: Install Add-on

  Go to https://metamask.io/

  Get chrome extension > add to chrome > add extension > done.

- Step 2: After Metamask is installed into Chrome Extension, it is able to use by clicking the “fox head” icon on the chrome toolbar. Click it!

  Users are now asked to enter a password by encrypting it, which protects their entry. It ensures more protection for users. Choose a password which is secure and easy to remember. Do not share anybody's password. Keep it in mind !

  Users now have 12 terms to reveal. This is a fancy private key underneath which produces an infinite amount of private keys. This is what users need to get to their Ether. No one can restore it if they lose it, die, reinstall Chrome, uninstall the Metamask extension or reinstall the OS. A better way is to store a second copy of these 12-word in a different physical location.

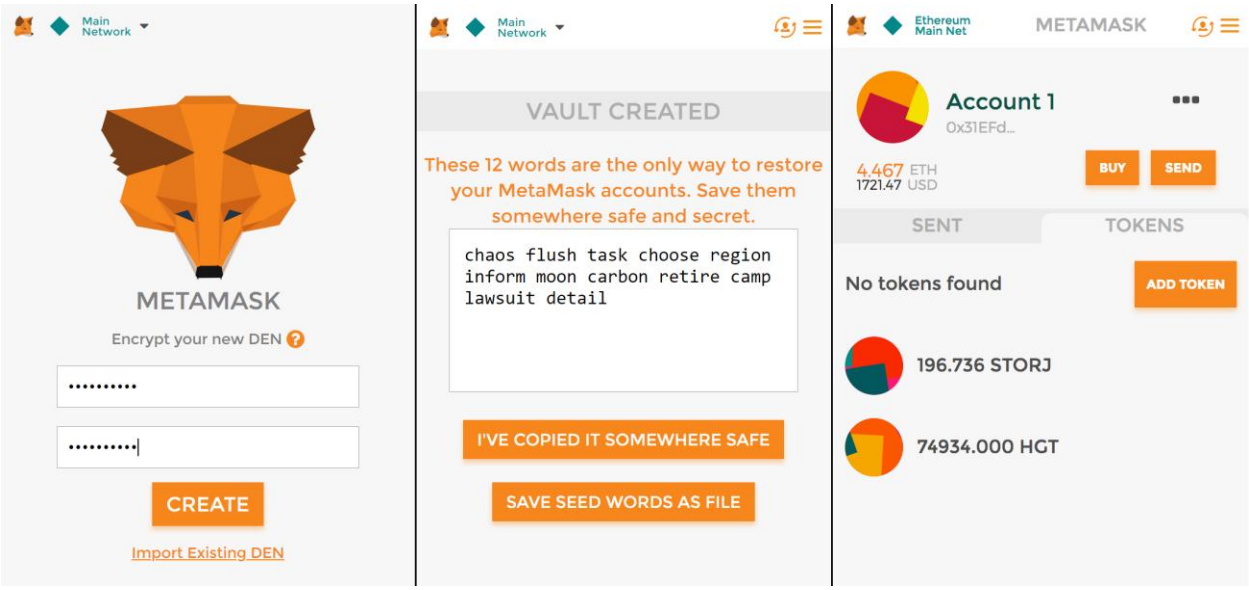

Figure 6.4. Metamask Creating Accounts and Back-Up Code

➢ Step 3: Customizing HTTP Provider by changing RPCs to different Ethereum Node to connect to the target Blockchain.

Chrome Extension API provides a home page for each extension. Please paste this to your browser's url panel to go to MetaMask home page.

chrome-extension://<Your-Chrome-Extension-ID>/home.html

Please paste this url chrome://extensions/ into your browser's url panel to see your MetaMask Extension's ID.

Now change the network to Ropsten Testnet to perform transactions in this thesis paper.

6.1.2.5. Truffle (Solidity Smart-Contract's Compiler and Deployer)

Truffle is Ethereum's development environment, testing platform, and asset pipeline, with the aim of making life easier as an Ethereum creator. You get with truffle:

- Built-in smart contract compilation, linking, deployment and binary management.
- Automated contract testing with Mocha and Chai.
- Configurable build pipeline with support for custom build processes.
- Scriptable deployment & migrations framework.
- Network management for deploying to many public & private networks.

- Interactive console for direct contract communication.
- Instant rebuilding of assets during development.
- External script runner that executes scripts within a Truffle environment.

Now it's time to use truffle ! Run the following command in prompt, and remember to change directory to this path folder <YOUR_PROJECT_FOLDER>/Real-Estate-API/.

➢ <npx truffle init --force>

NPX is a replacement for NPM command and it comes bundled with NPM version 5.2+ [11]. The only difference is that NPX takes local dependency inside node_modules instead of using global dependency. Please refer to part 2.1.1.2 - NPM to review about local packages and global packages.

The flag --force will empty all duplicated folder inside the current workspace, or the workspace that execute the NPX command, without prompting the warning.

6.1.2.6. Remix IDE with Remix Daemon

Let's first open a remix IDE, which is located at http://remix.ethereum.org/. Its provides a friendly interface for Ethereum Smart Contract developers.

Currently (at January 2020) there are 2 version of Remix IDE:

Version 0.7.7 is served at: http://remix.ethereum.org/#appVersion=0.7.7

Version 0.8.0 is served at: http://remix.ethereum.org/#appVersion=0.8.0

❖ In this thesis paper, I used Remix IDE Version 0.8.0 due to its good-looking UI, rather than the Version 0.7.7.

Following the steps specified in the **Error! Reference source not found.** below.

➢ Step 1: Changing the Remix's Theme, I used Cerulean (light) since this theme has good-looking interface to me.

➢ Step 2: Opening the Remix's Compiling Environment, by clicking Solidity and Vyper button in step 2.1. The UI will appear 2 environmental configurations for the 2 programming languages, which specified as buttons in step 2.2.

- Step 3: Connecting Remix IDE file system with your local file system. To be more specific, as you write your Solidity code in Remix, they are automatically saved in your local file system, the same where your local IDE is pointing to.

Now let's install remixd, which is a NPM module, as a global package on your computer. Its purpose is to give the remix web application access to a folder from your local computer. To be more specific, it synchronizes the local files in your computer to the remix web browser through a websocket port.

- <npm install remixd -g>

Please do not worry about the logged errors in Command Prompt. Now let's check all global dependencies installed in your computer.

- <cd contracts>

Finally, it's time to link your remix IDE in the browser to this local folder that stored Solidity Contracts, thanks to the help of remixd middleware. Continue to run the following command in the command prompt interface.

- <remixd -s ./ --remix-ide http://remix.ethereum.org>

Please make sure Remix IDE is running at http://remix.ethereum.org. If Remix IDE is running with Secure HTTP Protocol https://remix.ethereum.org, please change to HTTPS Protocol in the URL. Then run the following command in prompt with HTTPS URL.

- <remixd -s ./ --remix-ide https://remix.ethereum.org>

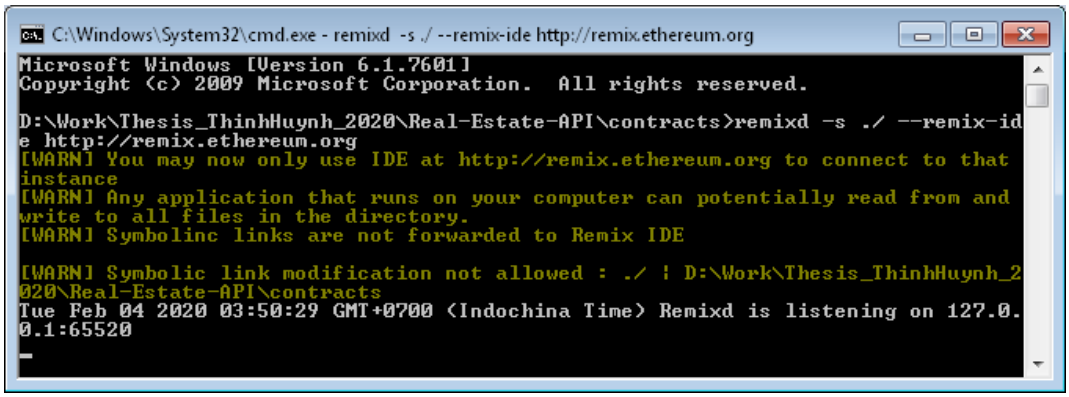

Figure 6.5. Remix Daemon with Web Socket

We have run a web socket server at ws://127.0.0.1:65520, and now let's connect Remix IDE with the local web socket by following step 3.1 then. Clicking "Connect to Localhost" button makes Remix prompts the below alert.

Migration.sol is the Solidity file that has been generated during Truffle initialization in part 6.1.2.5 - Truffle (Solidity Smart-Contract's Compiler and Deployer).

Now feel free to develop your smart contracts in Remix IDE, since any change in localhost file explorer in Remix IDE will synchronize with your local files.

6.1.2.7. Ganache (Private Testing Blockchain for Developers)

Ganache is a testing Ethereum Blockchain that is run locally on your computer, and it is used for fast deployment and testing Ethereum Smart Contracts.

There are 2 ways to install Ganache, you can use ganache-cli (Command Line Interface Ganache) or Ganache (Desktop Application).

The table below shows the comparison between 2 types of Ganache.

Table 6.1. Ganache CLI and Ganache Desktop UI

ganache-cli	Ganache Desktop Application
+ Fast start. – Manually save the used workspace after terminating the process. – Manually tracking the mined and pending transaction – Manually tracking the state of contracts by run the command in terminal line.	– Slow start. + Automatically save the workspace for later reused after terminating the process. + Automatically tracking and storing the mined and pending transactions. + Automatically tracking and storing the state of contracts

❖ **Install and Run Ganache-cli**

Now let's install Ganache Command Line Interface, which is a NPM module, as a global package on your computer. Its purpose is to imitate the Ethereum Blockchain Network that runs locally. To be more specific, it fakes the Blockchain Network for developers to deploy and test smart contracts without waiting the transactions to be mined. Run the following command.

- <npm install -g ganache-cli>

Now let's run ganache-cli with your Command Prompt by run the following command.

- <ganache-cli -p 7545>

Please reading Ganache CLI documents [12] to have more understanding about different arguments and flags that could be input through ganache-cli application.

But remember every time you run Ganache like this, it's going to create a totally fake new Ethereum Blockchain, means you could not save the current workspace. To save the workspace please run the following command in prompt.

- <ganache-cli -p 7545 --db <PATH-TO-YOUR-WORKSPACE>>

Since I used Windows 7, Ganache-CLI is installed globally in disk C, thus the path to your workspace must belongs to disk C. If the path belongs to disk C, Ganache-CLI will throw the error in the prompt after initiation failed ! So run ganache-cli with saving workspace options with the following command in prompt.

❖ Install and Run Ganache Desktop UI

Please download installation for Ganache Desktop at https://www.trufflesuite.com/ganache.

After installation, open Ganache and the UI appears as the Figure 6.6 below.

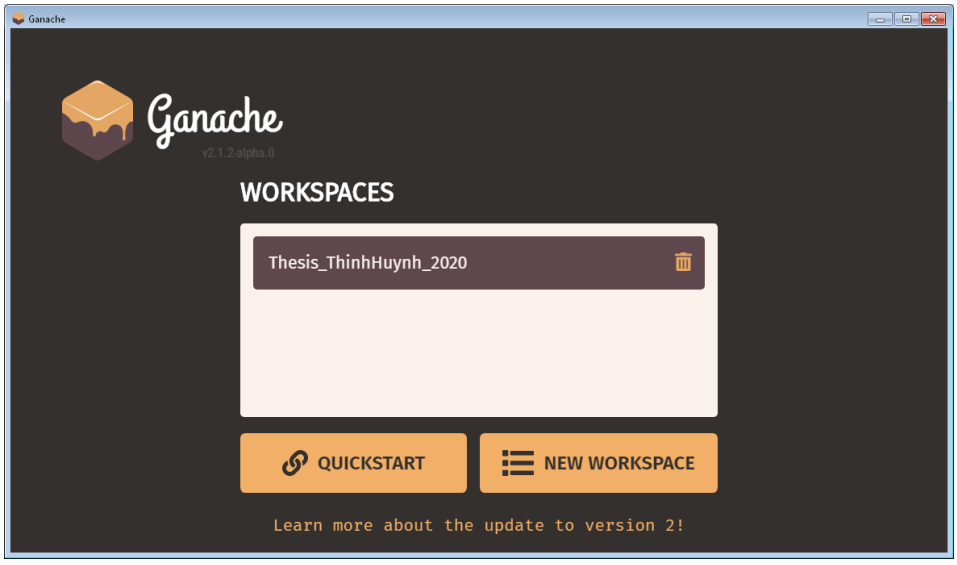

Figure 6.6. Ganache Home Page UI

In this thesis paper, I'll use port 7545 for Ganache, which means now I have a fake Ethereum Node that serves as a HTTP Provider for Web3 Application at http://localhost:7545/.

6.2. Implementation

In this part, I will neither show you the detail, nor else, fully explain the source code. But rather gives readers an abstract view of code structure and the functionality of each module or component specified in this thesis paper. Please visit my Github to access the fully documented paper and the source code at: https://github.com/minhthinhls/Thesis_ThinhHuynh_2019/.

6.2.1. Front-End / Client Side

The file structure inside <YOUR_PROJECT_FOLDER>/Real-Estate-CLIENT/ are illustrated as the Figure 6.7 below.

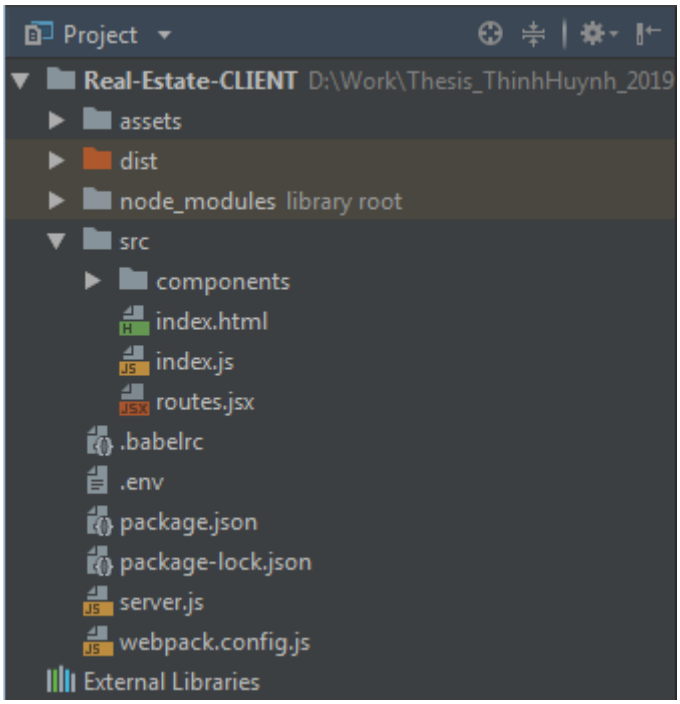

Figure 6.7. Project Structure - CLIENT

6.2.1.1. NPM Scripts in Client's Configuration.

Let first begin with NPM scripts specified in package.json in Real-Estate-CLIENT folder.

- <npm run dev>

The flags should be customized inside NPM scripts since the customized flag as <--mode> or <--hot> are sometime confused with the other developers.

6.2.1.2. Module Workflow in Client Side Architecture.

Client Side Service in this thesis paper is based on Server-Side Rendering, thus the Client requires a NodeJS platform to render HTML, CSS, JS before response rendered data to Client.

The Figure 6.8 below illustrates the workflow of bundling files using webpack and its supported modules by running the following command in prompt.

- **<npm run build>**

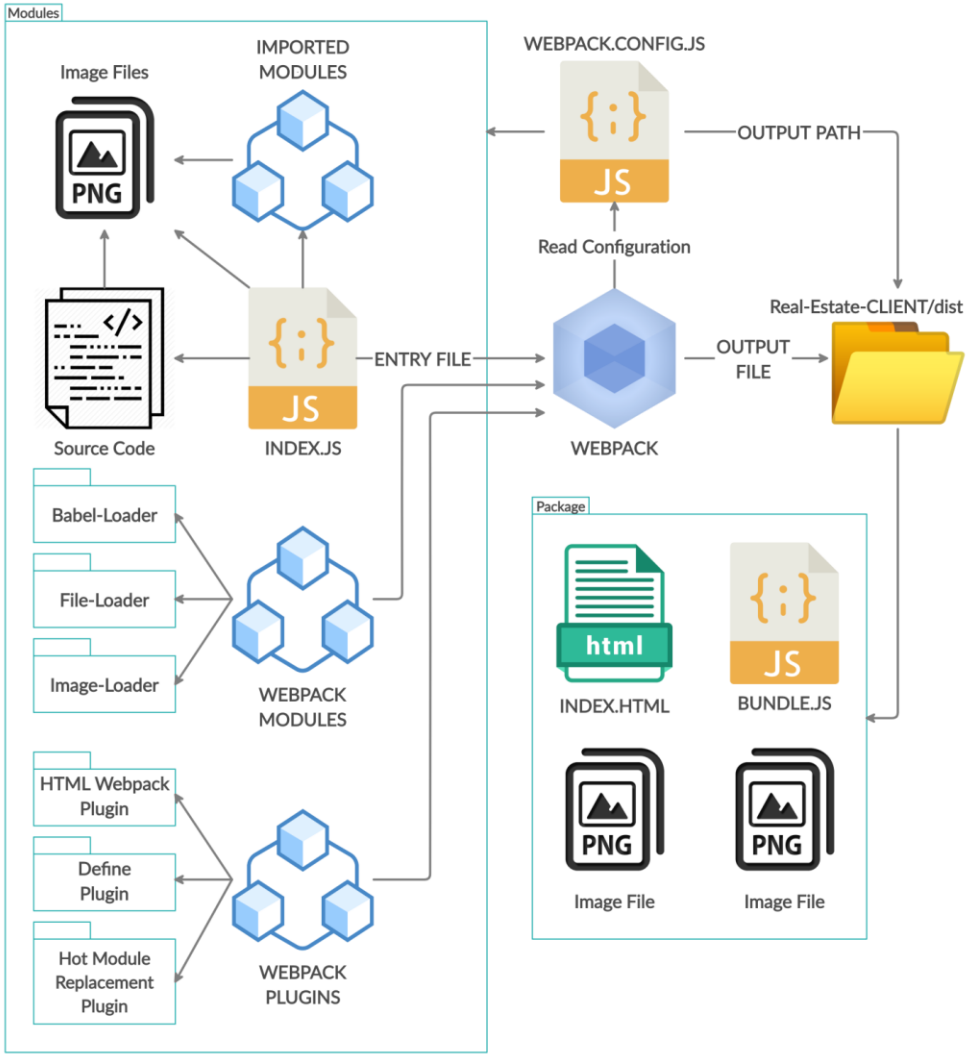

Figure 6.8. Webpack Bundling Files Workflow - <NPM Run Build>

The Figure 6.9 below illustrates the workflow of starting front-end rendering server using NodeJS with webpack and its supported modules by running the following command.

- **<npm run start>**

Please specify the Node Environment in .env file before deploy project to the server. For DEVELOPMENT, the code will automatically restart if Webpack Hot Module detects changes.

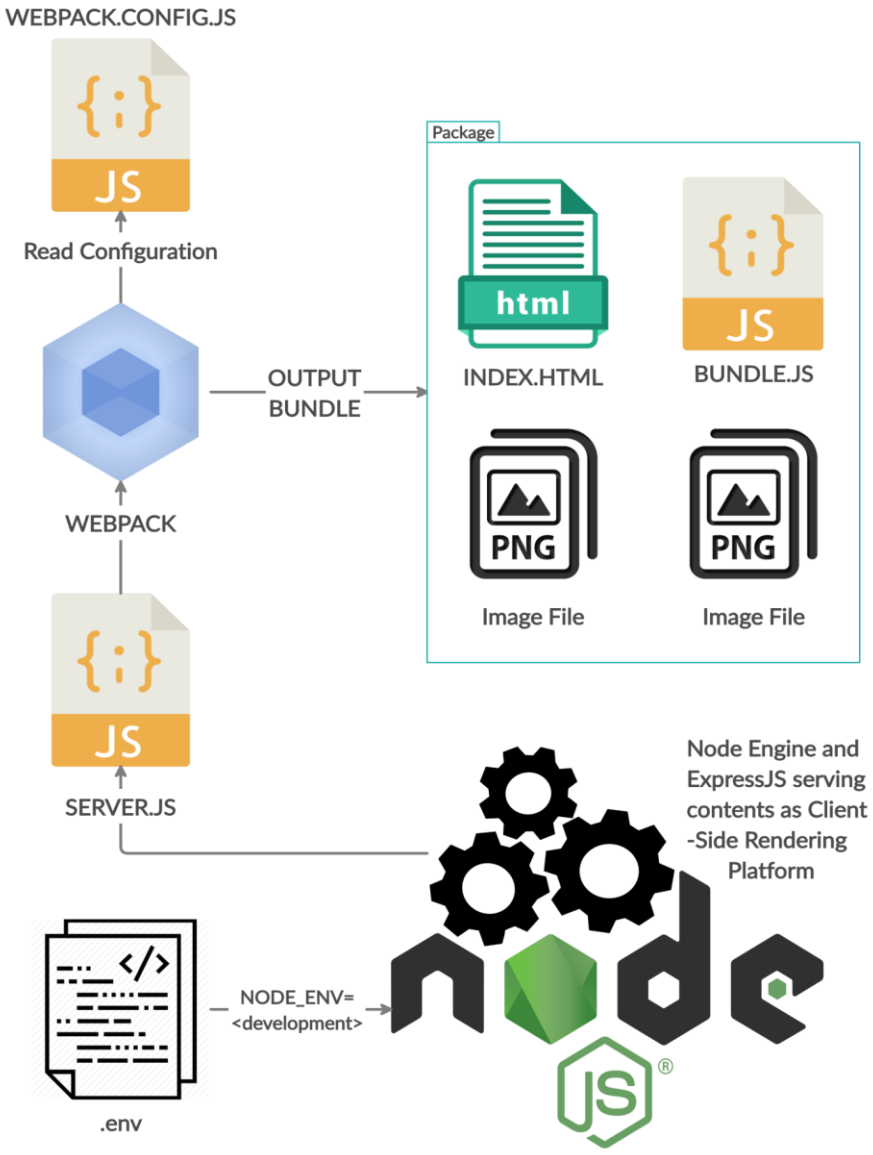

Figure 6.9. NodeJS Rendering Client Side for Development - <NPM Run Start>

For PRODUCTION environment, the code will not automatically restart after deployment, since Webpack Hot Middleware and Webpack Dev Middleware are not used in SERVER.JS file.

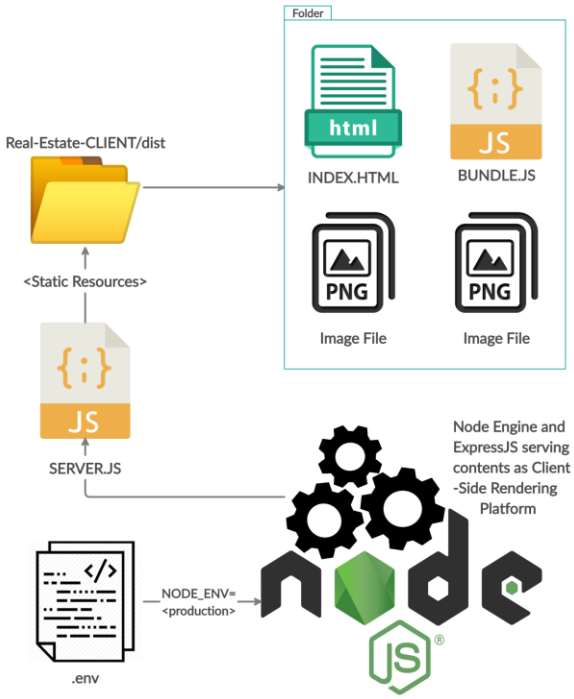

Figure 6.10. NodeJS Rendering Client Side for Production - <NPM Run Start>

- **<npm run dev>**

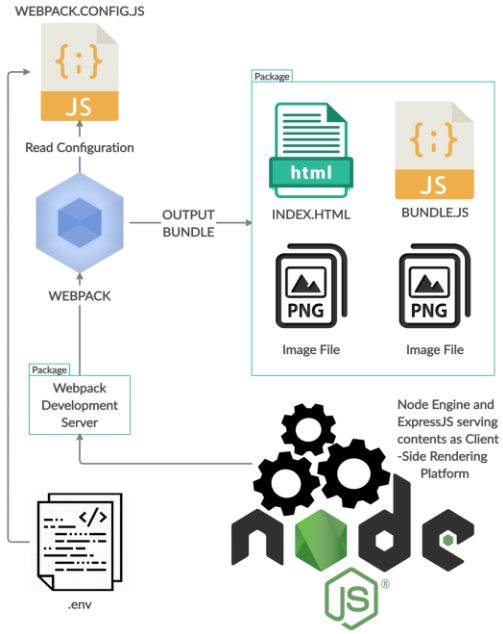

Figure 6.11. NodeJS Rendering Client Side for Development - <NPM Run Dev>

The Figure 6.11 above illustrates the workflow of starting front-end rendering server using NodeJS with webpack-dev-server by running the command <npm run dev>.

6.2.2. Back-End / Server Side

The file structure inside <YOUR_PROJECT_FOLDER>/Real-Estate-API/ are illustrated as the Figure 6.12 below.

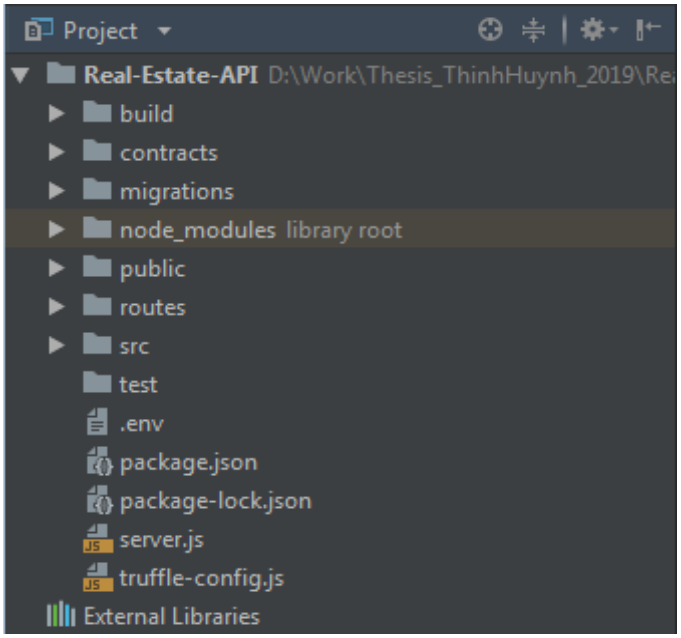

Figure 6.12. Project Structure - API

6.2.2.1. NPM Scripts in Server's Configuration.

Let first begin with NPM scripts specified in package.json in Real-Estate-API folder.

- <npm run deploy>

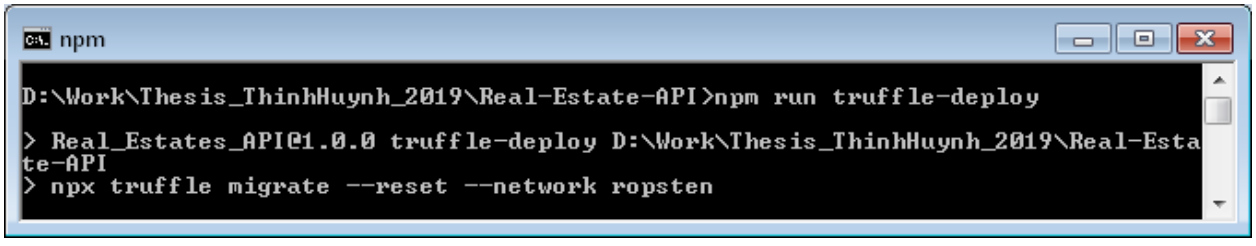

Figure 6.13. NPM Run Truffle Deploy - API

6.2.2.2. Module Workflow in Server Side Architecture.

Server Side Service in this thesis paper is based on Express and its Middlewares, thus the Server requires a NodeJS platform to apply Routing and Middlewares before responding to requests from Client Side.

The Figure 6.14 below illustrates the workflow of ExpressJS Server based on NodeJS platform and its wrapper or supported modules - Nodemon. This allow the ExpressJS Server could be automatically restarted if any things changed by changing Node Environment in .env folder to Nodemon, NODE_ENV="nodemon" and then run the following command in prompt.

- **<npm run nodemon>**

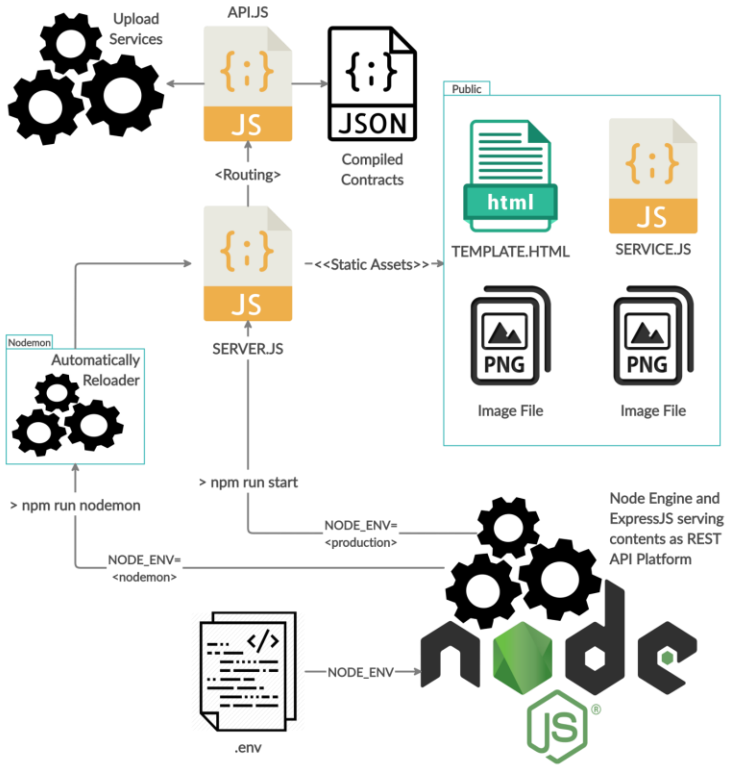

Figure 6.14. ExpressJS API Workflow - <NPM Run Nodemon>

Please keep in mind that in Production Environment, the server will not automatically restart if anything inside the project folder changes.

6.2.2.3. ExpressJS Code Structure and Router.

The implementation of this thesis paper is web-based platform, thus the API requires a file to manage the Express Server, which is server.js inside the Real-Estate-API folder.

The main flow of ExpressJS API follows the code structure that firstly starts with server.js file implemented as the Figure 6.15 below.

```
/* SERVER FILE TO RUN NODE ENVIRONMENT */
require('dotenv').config(); // Read <.env> file into $<process.env> global
variable.
const path = require('path');
const express = require('express');
const createError = require('http-errors');
const cookieParser = require('cookie-parser');
const logger = require('morgan');
const cors = require('cors');
const app = express();

/* All engine setup */
app.use(cors());
app.use(logger('dev'));
app.use(cookieParser());
app.use(express.json());
app.use(express.urlencoded({extended: false}));

/* Public all file within folder ./public and ./dist */
app.use('/dist', express.static(path.join(__dirname, 'dist')));
app.use('/public', express.static(path.join(__dirname, 'public')));

/* All router setup */
app.use('/api', require('./routes/api'));

/* Catch 404 and forward to error handler */
app.use(function (req, res, next) {
  next(createError(404));
});

/* Error handlers */
```

```
app.use(function (err, req, res) {
  /* Set locals, only providing error in development */
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  /* Render the error page */
  res.status(err.status || 500);
  res.render('error');
});

/* Finally, let's start our server... */
var server = app.listen(8080, function () {
  console.log('Listening on port ' + server.address().port);
});

module.exports = app;
```

Figure 6.15. Server File for ExpressJS API

The ExpressJS Application takes advantages of using middlewares by calling the function named <use> from the instance <app> that is generated by express dependency. Please review part 2.1.3.1 to understand the main flow of an Express Application.

Let's look into the setup of Express Router, the Figure 6.16 below illustrates the detail of services that is served by the Express API.

```
const express = require('express');
const router = express.Router();
const uploader = require('../src/js/uploader');

/* Application Programming Interfaces */
router.get('/contract/house', (req, res) => {
    res.json(require('../build/contracts/House.json'));
});

router.get('/contract/houseAdmin', (req, res) => {
    res.json(require('../build/contracts/HouseAdmin.json'));
});
```

```
// Service to receive Image Uploaded from Clients.
router.post('/upload', (req, res) => {
    uploader(req, res, (err) => {
        console.log("Request ---", req.body);
        console.log("Request file ---", req.file); // Here you get file.
        /* Now do where ever you want to do */
        if (!err) {
            return res.send(200).end();
        }
    })
});

module.exports = router;
```

Figure 6.16. ExpressJS Router for API Services

In this thesis paper, the implemented ExpressJS API could handle the uploading of Images from Clients and response to Clients the compiled Ethereum Smart Contracts.

6.2.3. Ethereum Smart Contracts

In this thesis paper, Truffle is applied to handle the compilation and deployment for Ethereum Smart Contracts. Please review part 6.1.2.5 - Truffle (Solidity Smart-Contract's Compiler and Deployer) to understand the basic concepts of Truffle.

Inside the Real-Estate-API folder, there's a file named truffle-config.js that's been generated under the command <npx truffle init> at Real-Estate-API folder. Here are the alternative path to the configuration file: <YOUR_PROJECT_FOLDER>/Real-Estate-API/truffle-config.js.

The Figure 6.17 below illustrates the configuration for Truffle in truffle-config.js file.

```
require('dotenv').config(); // Read <.env> file into $<process.env> global
variable.
const HDWalletProvider = require('truffle-hdwallet-provider');
const MNEMONIC = process.env.MNEMONIC;
const INFURA_KEY = process.env.INFURA_KEY;

module.exports = {
  networks: {
    development: {
```

```
      host: "127.0.0.1",
      port: 7545,
      network_id: "*",
    },
    ropsten: {
      provider: () => new HDWalletProvider(MNEMONIC, process.env.ROPSTEN +
INFURA_KEY),
      network_id: 3,
      gas: 5500000,
      confirmations: 2,
      timeoutBlocks: 200,
      skipDryRun: true
    }
  },
  mocha: {
    timeout: 100000
  },
  compilers: {
    solc: {
      version: "0.5.1",
      docker: true,
      settings: {
       optimizer: {
         enabled: false,
         runs: 200
       },
       evmVersion: "byzantium"
      }
    }
  }
};
```

Figure 6.17. Truffle Configuration File

The configuration contains 3 main part, which are: network, mocha and compiler.

- Network: I've already specified the development network at localhost, port 7545 (running Ganache) for development purposes and ropsten test network for demonstration.

- ❖ Mocha: The configuration for testing smart contract. To perform unit test please run the following command <npx truffle test --network development> in folder Real-Estate-API.
- ❖ Compiler: Compiling configuration provided by Truffle, developers could specify the version and optional customization for settings.

The configuration contains 3 main part, which are: network, mocha and compiler.

6.2.3.1. Compile Contracts

Since you have already done part 6.1.2.6 - Remix IDE with Remix Daemon. You can see all the Solidity contracts inside <YOUR_PROJECT_FOLDER>/Real-Estate-API/contracts.

To compile all Solidity smart contracts please run the following command in prompt at the Real-Estate-API directory. Fully path: <YOUR_PROJECT_FOLDER>/Real-Estate-API/.

- ➢ <npm run compile>

Or you can run the command directly with NPX for Truffle Compilation without using NPM Scripts replacement.

- ➢ <npx truffle compile --all>

Now all of your Solidity Contracts are compiled into JSON Contracts at the path folder: <YOUR_PROJECT_FOLDER>/Real-Estate-API/build/contracts.

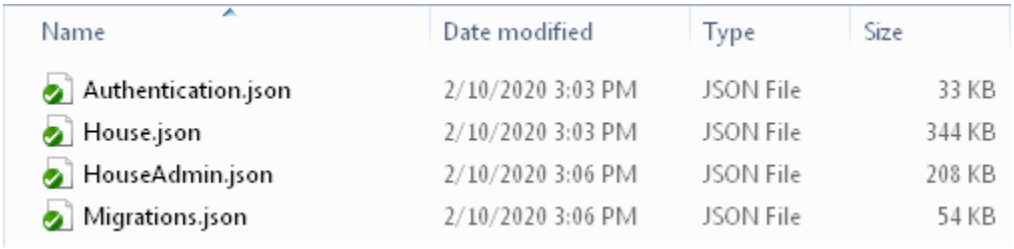

Figure 6.18. Compiled Contracts Folder

6.2.3.2. Deploy Contracts

Please review part 5.2.1 - NodeJS Server as Contract Deployer to have an overview of the system workflow to deploy Solidity Contract to Ethereum Network using Infura HTTP Provider.

The Figure 6.19 below illustrates the main flow of deploying the Compiled Solidity Contracts to either Ropsten Test Network or Developing Network (Ganache).

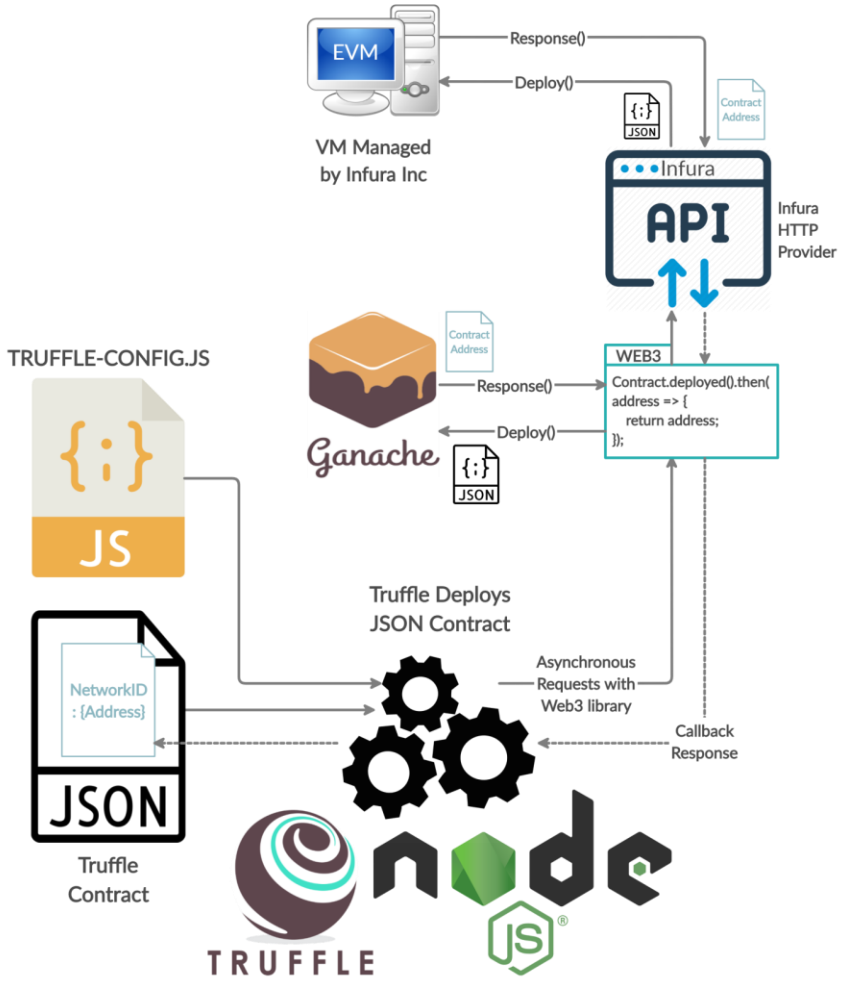

Figure 6.19. Truffle Deploys Contracts Workflow

Deploying Solidity Smart Contracts require following these steps in command prompt at the Real-Estate-API directory. Fully path: <YOUR_PROJECT_FOLDER>/Real-Estate-API/.

❖ Since HouseAdmin Contract must be deployed to the Ethereum Network, please modifying the JavaScript file named < 2_deploy_contracts.js> inside the alternative path folder: <YOUR_PROJECT_FOLDER>/Real-Estate-API/migrations/ as Figure 6.20 below.

```
var HouseAdmin = artifacts.require("./HouseAdmin.sol");

module.exports = function(deployer) {
  deployer.deploy(HouseAdmin);
};
```

Figure 6.20. Truffle Deployer Configuration

❖ To deploy Smart Contracts specified in < 2_deploy_contracts.js> to your development Blockchain (Ganache at localhost:7545), please run the following command in prompt.

➢ <npx truffle migrate --reset --network development>

However, deploying the House Admin Contract to Ganache is for development purpose only, as CHAPTER 3 - System Requirements has already specified that for thesis demonstration, the Ethereum Ropsten Test Net must be applied.

❖ To deploy Solidity Smart Contracts specified inside < 2_deploy_contracts.js> to the Ropsten Test Network Blockchain, please run the following command in prompt.

➢ <npx truffle migrate --reset --network ropsten>

Let's just wait for about 2 minutes since deploying to the Ropsten Test Network takes time.

6.2.3.3. Contract's Interaction

Interacting with Ethereum Blockchain Network requires Web3 dependency injected into the browser. Let's modify the source code of index.js in Client Side, see **Error! Reference source not ound.**. Fully path to JavaScript Index File: <YOUR_PROJECT_FOLDER>/Real-Estate-CLIENT/src/index.js

```
import React from 'react';
import {render} from 'react-dom';
import Routes from './routes';

const Web3 = require('web3');

try {
  if (typeof web3 !== 'undefined') {
    // If a web3 instance is already provided by MetaMask Browser Extension.
    window.web3 = new Web3(web3.currentProvider);
  } else {
    /* Injecting new Web3 instance with customized HTTP Provider at
http://localhost:7545 if no web3 instance provided */
    window.web3 = new Web3(
      new Web3.providers.HttpProvider('http://localhost:7545')
    );
  }
```

```
  web3.eth.defaultAccount = web3.eth.accounts[0]; // Set first accounts as
default one !
} catch (Exception) {
  console.log(Exception);
}

try {
  /* Add Event Listeners for MetaMask Browser Extension */
  window.ethereum.on('accountsChanged', function (accounts) {
    web3.eth.defaultAccount = accounts[0];
    render(<Routes/>, document.getElementById('app'));
  });
  window.ethereum.on('networkChanged', function (networkID) {
    web3.version.network = networkID;
    render(<Routes/>, document.getElementById('app'));
  });
} catch (Exception) {
  console.log(Exception);
}

render(<Routes/>, document.getElementById('app'));

if (module.hot) {
  module.hot.accept();
}
```

Figure 6.21. JavaScript Index File - CLIENT

If the browser detects a web3 instance that is injected during the opening of MetaMask Extension, then we should reuse that web3 instance. On the other hand, a new web3 instance must be created in case the user hasn't installed and opened MetaMask.

MetaMask also support Event Listener API for developers. Since the application must be re-rendered after the users change their accounts or change the network, I have already put 2 event listeners, namely: "accountsChanged" and "networkChanged" using "on" function provided by MetaMask Programming API.

CHAPTER 7. CONCLUSION & FUTURE DISCUSSION

7.1. Summary

7.1.1. Work done

The 5 figures below shows the implementation of designed User Interfaces as in part 5.1.1 - User Interface (UI).

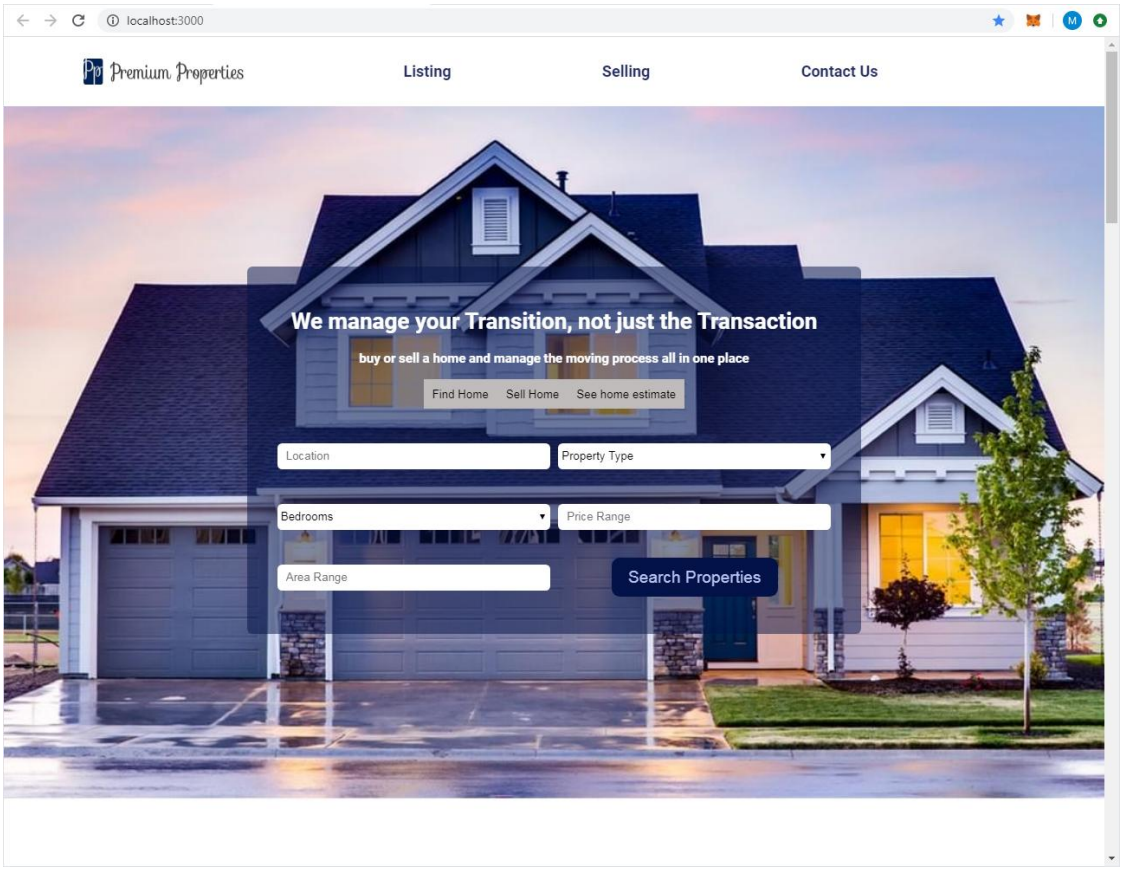

Figure 7.1. Home Page

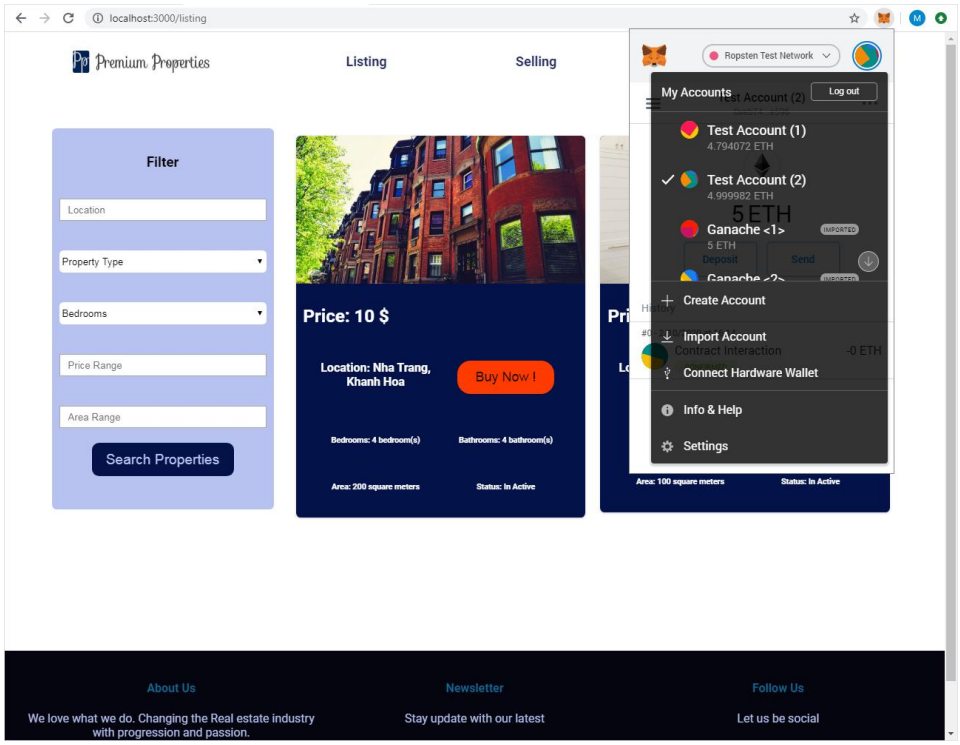

Figure 7.2. Listing Page

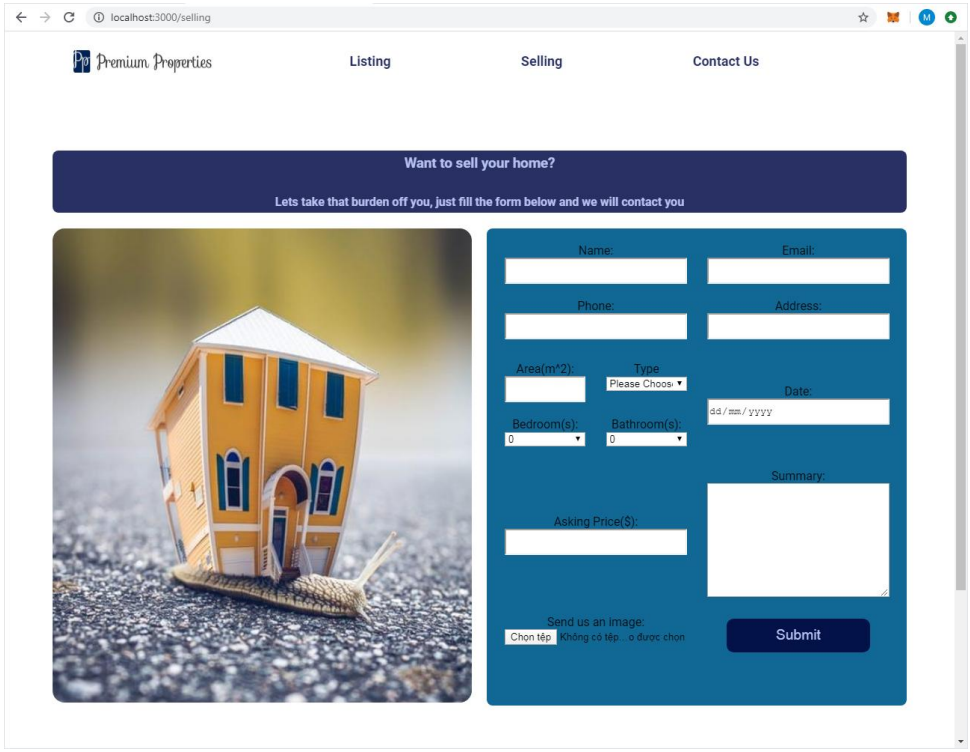

Figure 7.3. Sell Page

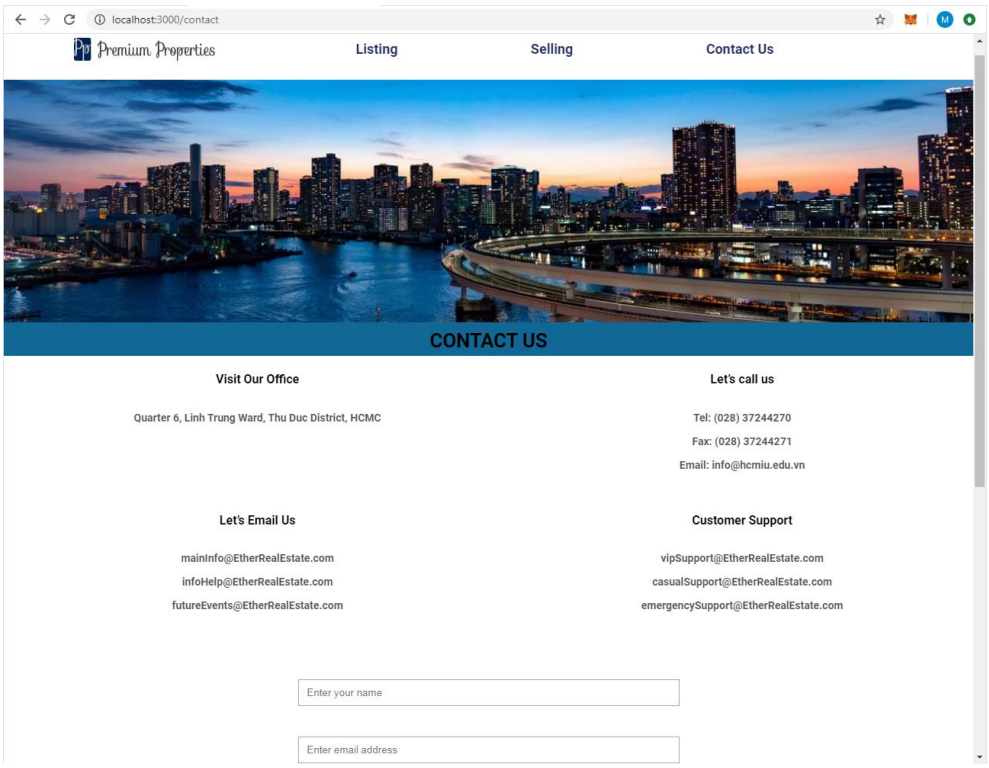

Figure 7.4. Contact Page

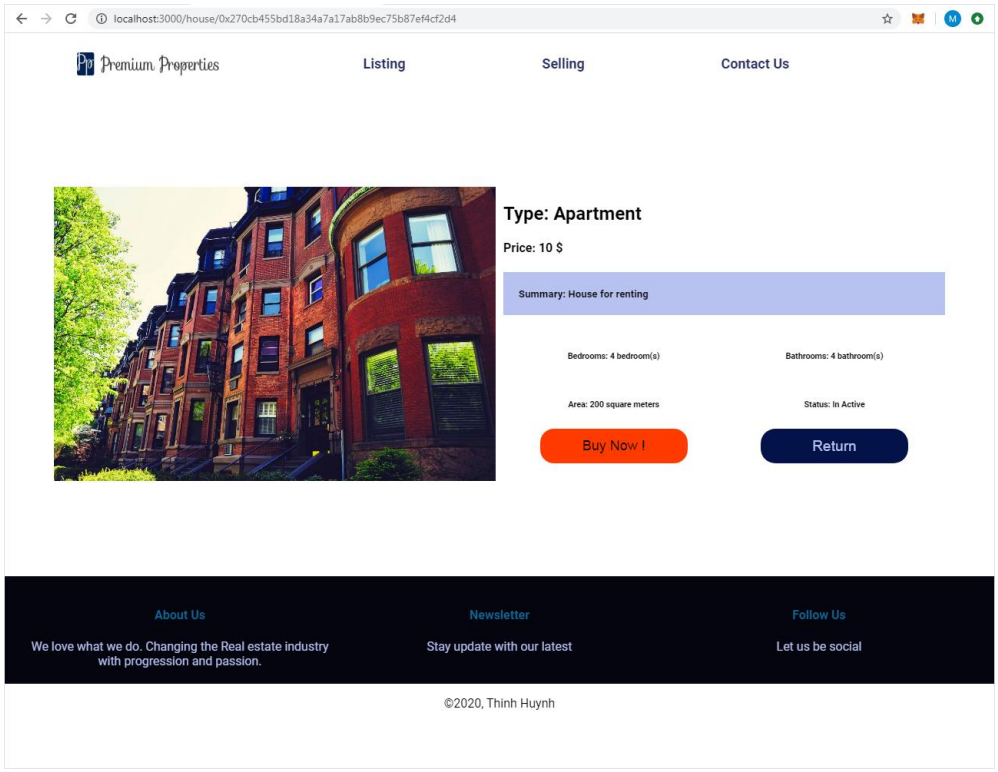

Figure 7.5. House Detail Page

The User Interfaces are friendly and simple for users to manually buying and selling their real-estate properties.

7.1.2. Conclusion

This thesis paper has already showed the combination between Web2.0 and Web3.0 technology. However, with Ethereum Blockchain Integration into Web2.0 technology even makes the system more reliable and transparent. But how Ethereum Blockchain secures the transactions that are made from users ?

❖ Once the transactions are mined into the chain of blocks by miners, there's no way to tamper the data even if the attackers are miners. This prevent a transaction to be mined twice (prevent double-spending attack), thus blocking the sellers to sell their house many times !

❖ Moreover, Ethereum Platform provides Solidity Smart Contracts for developers and users. Are smart contracts trustful ? The answer is totally yes, because they are stored internally on every EVM and managed by transactions stored in Ethereum Blockchain System. Hence, the Solidity Contracts inherit some sophisticated properties.

- Firstly, they are immutable and distributed. Once a smart contract is created, it can never be changed. Hence, no one could go behind the back of the chain of blocks and edit the code of a smart contract.
- Secondly, the output of a smart contract in a blockchain system is validated by every full-node on the network, which means nobody could force one smart contract to release a wanted output because other nodes on the network will spot this attempt and mark it as invalid. This ability is related to smart contract distributed property.

In conclusion, tampering with smart contracts becomes almost impossible with Ethereum Platform. And this eliminate the uses of third party programs for security as in Web2.0 technology, since the Blockchain System itself provides the security due to consensus between each node.

7.2. Future work (Promising improvements)

This thesis paper is based on both Centralized (Web2.0) and Decentralized (Web3.0) infrastructure. However, in the near future, an upgraded version of Real-Estate-Application with fully decentralized system will be released. Taking advantages of Swarm Decentralized Hosting

and IPFS Decentralized File Services. See Figure 2.34 to have an overview of how a Fully Decentralized Application structured.

Blockchain is a promising land for new developers who want to commit their abilities to the community by solving difficult puzzles that can only be adaptive with Blockchain Technology.

Furthermore, in the future some experts predict that the large enterprises will build their own personal blockchain networks. According to the report of the enterprise project, “Suppliers and customers cannot and would not join the private blockchain for every one of their business partners”. And, “The long-term future of the blockchain relies on the ability of companies in order to conduct private business over a public.”

LIST OF REFERENCES

Bibliography

[1] *Overview of Blocking vs Non-Blocking*. URL: https://nodejs.org/en/docs/guides/blocking-vs-non-blocking (visited on 11/30/2019)

[2] *The JavaScript Runtime Environment*. URL: https://medium.com/@olinations/the-javascript-runtime-environment-d58fa2e60dd0 (visited on 11/30/2019)

[3] *Introduction to JavaScript Engine: Call Stack, Event Loop, and Task Queue*. URL: https://medium.com/jsninja/call-stack-event-loop-and-task-queue-d49eff2ec7bb (visited on 11/30/2019)

[4] *Introduction to NPM*. URL: https://blog.cloudboost.io/introduction-to-npm-d62b6f6efd08 (visited on 11/30/2019)

[5] *What is the Ethereum Virtual Machine ?* URL: https://www.bitrates.com/guides/ethereum/what-is-the-unstoppable-world-computer (visited on 12/30/2019)

[6] *How does Ethereum Virtual Machine work ?* URL: https://medium.com/mycrypto/the-ethereum-virtual-machine-how-does-it-work-9abac2b7c9e (visited on 12/30/2019)

[7] *Getting Deeply Into EVM - How does Ethereum Virtual Machine work ?* URL: https://hackernoon.com/getting-deep-into-evm-how-ethereum-works-backstage-ac7efa1f0015 (visited on 12/30/2019)

[8] *MetaMask Wallet Review*. URL: https://www.bitdegree.org/tutorials/metamask-wallet-review/ (visited on 12/30/2019)

[9] *How to Customizing Web3 Http Provider ?* URL: https://ethereum.stackexchange.com/questions/41783/what-are-the-web3-httpprovider-options (visited on 12/30/2019)

[10] *What is Hierarchical Deterministic Wallet (HD Wallet)*. URL: https://tiendientu.org/hd-wallet-la-gi (visited on 01/30/2020)

[11] *Difference between npx and npm ?* URL: https://stackoverflow.com/questions/50605219/difference-between-npx-and-npm (visited on 01/30/2020)

[12] *Ganache CLI Configuration and Its Usability*. URL: https://docs.nethereum.com/en/latest/ethereum-and-clients/ganache-cli (visited on 01/30/2020)

[13] *Introduction to React Children*. URL: https://viblo.asia/p/tim-hieu-ve-children-trong-react-oOVlYqWol8W (visited on 01/30/2020)

Figure Citation

[1] *Web2.0 System with REST API*. URL: https://movan.vn/rest-api-gioi-thieu-rest-api/ (visited on 11/30/2019)

[2] *NodeJS Runtime Environment - Full Explanations*. URL: https://medium.com/@olinations/the-javascript-runtime-environment-d58fa2e60dd0 (visited on 11/30/2019)

[3] *NodeJS Runtime Environment - Simple*. URL: https://blog.sessionstack.com/how-javascript-works-event-loop-and-the-rise-of-async-programming-5-ways-to-better-coding-with-2f077c4438b5 (visited on 11/30/2019)

[4] *Babel Transpiler I/O*. URL: https://babeljs.io/docs/en/ (visited on 11/30/2019)

[5] *How Webpack Works*. URL: https://www.cleveroad.com/blog/gulp-browserify-webpack-grunt (visited on 12/30/2019)

[6] *Introduction to ExpressJS*. URL: https://medium.com/@provish/backend-for-frontend-43053e1fa3 (visited on 12/30/2019)

[7] *ExpressJS Workflow*. URL: https://www.ubuntupit.com/best-nodejs-frameworks-for-developers/ (visited on 12/30/2019)

[8] *Fundamental Technologies of Blockchain*. URL: https://www.paxos.com/the-blockchain-is-evolutionary-not-revolutionary-e80cb04e07ee/ (visited on 12/30/2019)

[9] *Hash Function*. URL: https://www.reply.com/solidsoft-reply/en/content/blockchain-an-overview (visited on 01/30/2020)

[10] *Transactions Merkle Tree*. URL: https://medium.com/coinmonks/merkle-made-palatable-94e6662f4caf (visited on 01/30/2020)

[11] *Semantic Bitcoin Block Components*. URL: https://www.researchgate.net/figure/Bitcoin-block-structure_fig1_335832820 (visited on 11/30/2019)

[12] *Bitcoin Blockchain Structure*. URL: https://medium.com/coinmonks/https-medium-com-ritesh-modi-solidity-chapter1-63dfaff08a11 (visited on 11/30/2019)

[13] *Miners Collect Transactions & Proof-of-work Mechanism*. URL: https://www.researchgate.net/publication/324078091_Dietcoin_shortcutting_the_Bitcoin_verification_process_for_your_smartphone (visited on 11/30/2019)

[14] *Blockchain Mining Mechanism*. URL: https://www.researchgate.net/figure/To-add-a-new-transaction-to-the-current-blockchain-a-miner-first-verifies-the-validity_fig1_324078091 (visited on 11/30/2019)

[15] *Abstract Keypair Generator*. URL: https://commons.wikimedia.org/wiki/File:Public_key_making.svg (visited on 12/30/2019)

[16] *Asymmetric Keypair with Proof-of-identity*. URL: https://medium.com/@blairlmarshall/how-does-a-bitcoin-transaction-actually-work-1c44818c3996 (visited on 12/30/2019)

[17] *Asymmetric Keypair with Encryption and Decryption*. URL: https://hackernoon.com/generating-rsa-private-and-public-keys-b82a06db6d1c (visited on 12/30/2019)

[18] *Ethereum Platform*. URL: https://blockgeni.com/product-managers-guide-to-blockchain/ (visited on 12/30/2019)

[19] *EVM's Simple Stack-Based Architecture & Execution Model*. URL: https://www.bitcoinisle.com/2018/07/23/tron-trx-virtual-machine-vs-ethereum-eth-vm-upgraded-tvm-launch-on-30th-july/ (visited on 12/30/2019)

[20] *Unit Conversion in Ethereum Blockchain*. URL: https://ethgasstation.info/blog/gwei/ (visited on 01/30/2020)

[21] *Solidity Smart Contract*. URL: https://medium.com/coinmonks/compiling-and-deploying-ethereum-smart-contracts-with-pure-javascript-4bee3bfe99bb (visited on 01/30/2020)

[22] *Ethereum Smart Contract Interaction*. URL: https://www.researchgate.net/figure/The-process-of-smart-contracts-development-deployment-and-interaction_fig3_319249505 (visited on 01/30/2020)

[23] *Interaction Between Accounts and Blockchain network*. URL: https://hackernoon.com/product-managers-guide-to-blockchain-part-3-fb0cffbff7f8 (visited on 01/30/2020)

[24] *Core Components of Ethereum Decentralized Applications*. URL: https://medium.com/coinmonks/getting-started-with-ethereum-and-building-basic-dapp-ebb681fb3748 (visited on 01/30/2020)

[25] *Ethereum Fully Decentralized Applications*. URL: https://medium.com/blockchannel/tools-and-technologies-in-the-ethereum-ecosystem-e5b7e5060eb9 (visited on 01/30/2020)