Introduction

A Smart Contract also known as a crypto contract, is a computer program that directly controls the transfer of digital currencies or assets between parties under certain conditions. A smart contract not only defines the rules and penalties around an agreement in the same way that a traditional contract does, but it can also automatically enforce those obligations. It does this by taking in information as input, assigning value to that input through the rules set out in the contract, and executing the actions required by those contractual clauses.

Smart contracts are complex and their potential goes beyond the simple transfer of assets, being able to execute transactions in a wide range of fields, from legal processes to insurance premiums to crowd funding agreements to financial derivatives. Smart contracts have the potential to disinter mediate the legal and financial fields, in particular, simplifying and automating routine and repetitive processes for which people currently pay lawyers and banks sizable fees to perform. The role of lawyers could also shift in the future as smart contracts gain traction, for example form adjudicating traditional contracts to producing customizable smart contract templates. Additionally, smart contracts' ability not only to automate processes but also to control behavior, as well as their potential in real time auditing and risk assessments, can be beneficial to compliance.

1.1 Project Motivation

Autonomy : You're the one making the agreement, there's no need rely on a broker, lawyer or other intermediates to confirm. Incidentally, this also knocks out the danger of manipulation by a third party, since execution is managed automatically by the network, rather than by one or more, possibly biased, individuals who may err.

Trust : Your document are encrypted on a shared ledger. There's no way that someone can say they lost it.

Backup : Imagine if your bank lost your savings account. On the Blockchain each and every one of your friends has your backup. Your documents are duplicated many times over.

Safety : Cryptography, the encryption of websites, keeps your documents safe. There is no hacking. In fact, it would take an abnormally smart hacker to crack the code and infiltrate.

Speed: You'd ordinarily have to spend chunks of time and paper work to manually process documents. Smart Contracts use software code to automate tasks, thereby saving hours off range of business processes.

Savings : Smart Contracts save you money since they knock out the presence of an intermediary. You would, for instance, have to pay a notary to witness your transaction.

Accuracy : Automated contracts are not only faster and cheaper but also avoid the errors that come from manually filling out heaps of forms.

Why Smart Contract ?

In an organization, computers are connected to each other which make a network. In the network, various tasks are completed by different computers and data is shared among computers. Every computer is controlled by different methods and different ways of processing are done on the network. On the network, some computers have high processing power as compared to others.

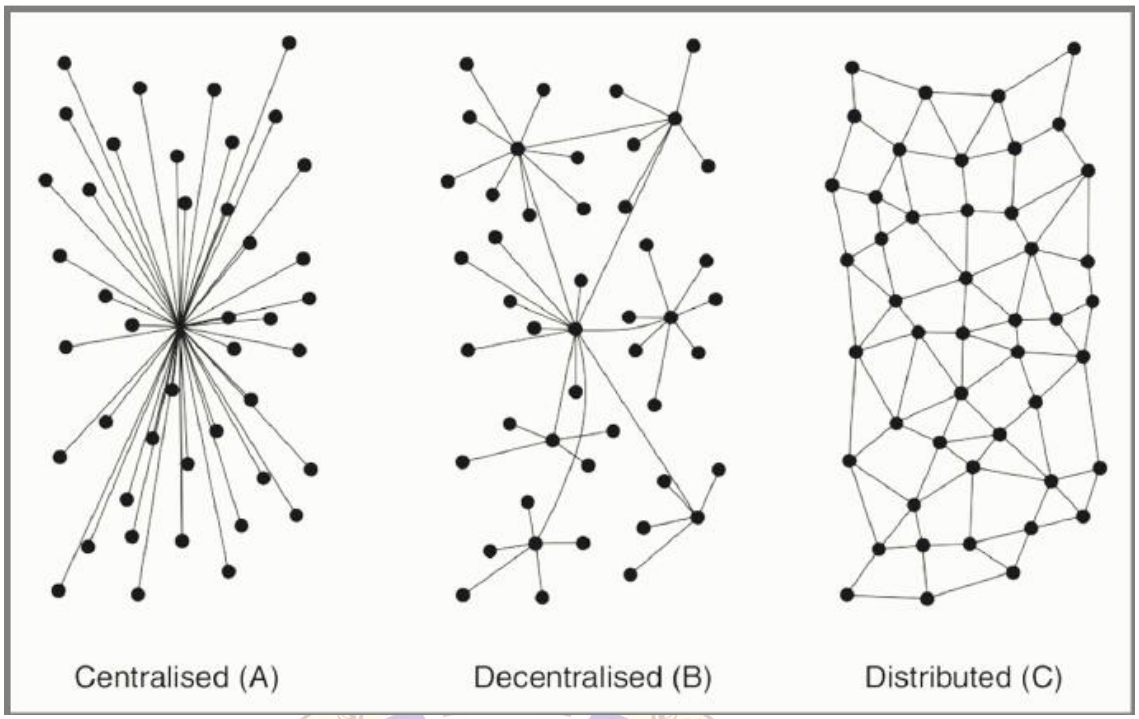


Figure 1.1 Centralised, Distributed and Decentralised [11]

In Centralized Processing, one or more terminals are connected to a single processor. Note that terminal is the combination of mouse, keyboard, and screen. In-library there is one processor attached to different terminals and library users can search any book from the terminal (mouse, keyboard, and screen). In centralized processing all the terminals are controlled by a single processor (CPU) and any command can be fulfilled by a single processor and this type of network is called centralized network.

In Decentralized Processing, there are different CPU connected on the network and each processor can do its job independent of each other. For example, in a Net cafe, all computers can perform their own tasks. This type of network is called decentralized network.

Another type of processing also exists named distributed processing. In this type of processing different CPU are connected to the network and are controlled by single CPU. For example in air reservation system there exists different terminals and processing is done from many locations and all the computers are controlled by the single main processor. This type of network is called distributed network.

Difference between Traditional Apps and Decentralized Apps

• The traditional web application uses HTML, CSS and JavaScript to render a page. It will also need to grab details from a database utilizing an API. When you go onto Facebook, the page will call an API to grab your personal data and display them on the page. Traditional websites: Front End → API → Database

• dApps are similar to a conventional web application. The front end uses the exact same technology to render the page. The one critical difference is that instead of an API connecting to a Database, you have a Smart Contract connecting to a Blockchain. dApp enabled website: Front End → Smart Contract → Blockchain

As opposed to traditional, centralized applications, where the backend code is running on centralized servers, dApps have their backend code running on a decentralized P2P network. Decentralized applications consist of the whole package, from backend to frontend. The smart contract is only one part of the dApp:

• Frontend (what you can see), and

• Backend (the logic in the background).

A smart contract, on the other hand, consists only of the backend, and often only a small part of the whole dApp. That means if you want to create a decentralized application on a smart contract system, you have to combine several smart contracts and rely on 3rd party systems for the front-end.

Applications of Smart Contract:

Blockchain and the potential to help notoriously difficult industries :

Whether you're starting a new job or buying a new phone, contracts are integral to any official agreement. The sheer volume and complexity of traditional contracts can be overwhelming, involving high administrative costs, dependence on a third party system and often outright confusion. As processes are increasingly digitalized, it's become necessary to find a way to make reliable, digital business agreements. Enter the smart contract, a computerized protocol which stores and carries out contractual clauses via Blockchain. The point is to avoid relying on third party systems, and allow visibility and access for all relevant parties. But what exactly can they be used for :

1. Insurance

Due to a lack of automated administration, it can take months for an insurance claim to be processed and paid. This is as problematic for insurance companies as it is for their customers, leading to admin costs, gluts, and inefficiency. Smart contracts can simplify and streamline the process by automatically triggering a claim when certain events occur. For example, if you lived in an area that was hit by a natural disaster and your house sustained damage, the smart contract would recognize this and begin the claim. Specific details (such as the extent of damage) could be recorded on the Blockchain in order to determine the exact amount of compensation. The same series of events would happen following a car accident, or if somebody reported an insured personal device as stolen.

2. Supply chain management

Supply chain management involves the flow of goods from raw material to finished product. Smart contracts can record ownership rights as items move through the supply chain, confirming who is responsible for the product at any given time. This has become far easier using Internet of Things sensors, which track goods from producers to warehouses, from warehouses to manufacturers, and from manufacturers to suppliers. The finished product can be verified at each stage of the delivery process until it reaches the customer. If an item is delayed or lost, the smart contract can be consulted to find out exactly where it should be. If any stakeholder fails to meet the terms of the contract, for instance if a supplier did not send a shipment on time, it would be clear for every party to see. Making supply chains more transparent via smart contracts is helping to smooth out the movement of goods and restore trust in trade.

3. Mortgage loans

The mortgage process is far from simple. The terms of a mortgage agreement, for example, are based on an assessment of the mortgagee's income, outgoings, credit score and other circumstances. The need to carry out these checks, often through third parties, can make the process lengthy and complicated for both the lender and the

mortgagee. Cut out the middle men, however, and parties could deal directly with each other (as well as access all the relevant details in one location). As a general rule, the simpler something is, the cheaper it will be – and through smart contracts, US lenders alone could reportedly save a minimum of $1.5bn.

4. Employment contracts

The relationship between an employee and their employer can be tempestuous, especially if either party fails to meet expectations. By entering into a smart contract, an employee would know exactly what was expected of them, as would the employer. Recording interactions in this way could help to improve fairness in wages or conditions, as any changes to contracts would be recorded. This openness could greatly improve the relationship between employers and their employees. Smart contracts could additionally be used to facilitate wage payments, according to the agreed amount and within a specific time period. Smart contracts could also help to regulate the use of temporary labor, which involves an employer, an agency and a worker. The worker joins the agency and is then hired by an employer. Unfortunately, a lack of transparency has meant that agencies can alter the contract's terms after workers have already started the job. This could mean shortening or lengthening the contract, changing wage rates or other worker's rights. It can be difficult for the authorities to detect these changes, but not if a smart contract system is applied.

5. Protecting copyrighted content

Every time that a piece of content is used for commercial purposes, for example a song, the owner of the rights to that song receives a royalty fee in theory. Of course, there are multiple parties involved in creating a song, and it can be hard to work out who owns these rights and who is therefore entitled to payment, plus existing systems do not work well. This has led to confusion over entitlement, no doubt giving some contributors more than they are due to the detriment of others while some receive nothing at all. Smart contracts can ensure that royalties go to the intended recipients by recording ownership rights in a decentralized Blockchain system. This could theoretically be applied to any piece of content with a team of contributors.

Smart contracts have many benefits for a wide range of industries, reducing unnecessary costs and time expenditure while enhancing transparency. In theory, they are more efficient and trustworthy than traditional contract law, and are also thought to offer better security as all actions are recorded and verified. However, like paper contracts, they could still experience fraud. Code is not infallible and can be delayed, intercepted and corrupted. As businesses move forward into digital negotiations, an awareness of these risks is integral.

Explanation with example :

Let us consider a polling app which is used to vote for candidates and let us consider we are making this in two ways :

1) Traditional app
2) Dapp

Traditional app will use basic backend to make app like java, python and lets say this is hosted on cloud server. But when we see today's scenario of cyber crime ,this app is hackable and data can be hacked easily .But instead of this if we make dapp this is claimed that around 500 super computers are needed to hack a dapp.

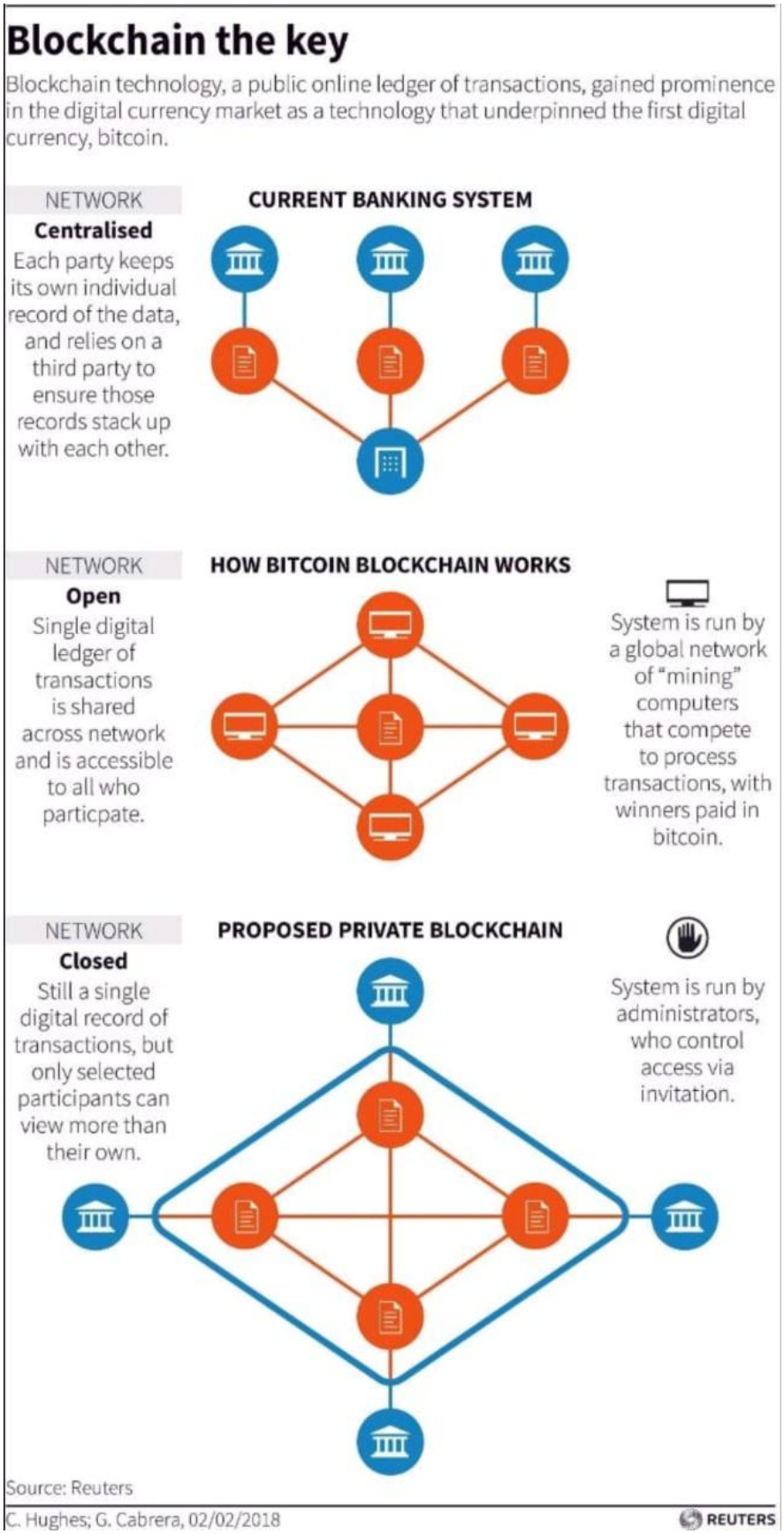


Figure 1.2 - Blockchain the Key [12]

1.2 Report Organisation

The rest of the report is organized as follows:

Chapter 1 gives a brief introduction to the concepts used in the report and organization of the report.

Chapter 2 describes the software requirements specifications (SRS) of the project.

Chapter 3 gives the literature review, all the work that has been done till this date and comparison of various algorithms.

Chapter 4 tells us about the system designing and the data flow concept through the software.

Chapter 5 gives the proposed solution to the problem and the mathematical modeling for solution and assumptions.

Then, the column defines the conclusion and future work for the project.

And, final document provides the references.

2.1. Introduction

2.1.1 Purpose

This document specifies the software requirements for the Smart Contract.

2.1.2 Document Conventions

The Smart Contract application is frequently referred to as “the application”.

2.1.3 Intended Audience and Reading Suggestions

This document describes the project scope for software developers. Readers should also be familiar with the Programming Languages under Windows platform.

2.1.4 Project Scope

A Smart Contract, also known as a crypto contract, is a computer program that directly controls the transfer of digital currencies or assets between parties under certain conditions. A smart contract not only defines the rules and penalties around an agreement in the same way that a traditional contract does, but it can also automatically enforce those obligations. It does this by taking in information as input, assigning value to that input through the rules set out in the contract, and executing the actions required by those contractual clauses.

2.2 Overall Descriptions

2.2.1 Product Perspective

Smart Contract application leverages the features of Remix Solidity and Web3.js to create a comprehensive interface. The Strength of the project should lie in the implementation and cohesion between its individual parts. These parts work together to create a proven, high performance application to controls the transfer of digital currencies or assets between parties under certain conditions.

2.2.2 Product Features

The features of the application fall into the following divisions:

This Project uses Go Ethereum Protocol and Mist Wallet to build smart contract by using Remix Solidity Version 0.4.0. Smart Contract is build between multiple wallet with at least two author's who are responsible for authorizing the contract among various parties.

A Smart Contract not only defines the rules and penalties around an agreement in the same way that a traditional contract does, but it can also automatically enforce those obligations.

2.2.3 Operating Environment

The application is to be developed in JetBrains PhpStorm 2017.3.4, Go Ethereum Prorocol and Mist Wallet using Remix Solidity programming language. It runs on Window 8 and Window 10 .

2.3 APPLICATON ENVIORNMENT

2.3.1 TECHNOLOGY USED

2.3.1.1 ETHEREUM

Ethereum is an open-source, public, blockchain-based distributed computing platform and operating system featuring smart contract (scripting) functionality. It supports a modified version of Nakamoto consensus via transaction based state transitions. In popular discourse, the term *Ethereum* is often used interchangeably with *Ether* to refer to the cryptocurrency that is generated on the *Ethereum platform.* Ether is a cryptocurrency whose blockchain is generated by the Ethereum platform. *Ether* can be transferred between accounts and used to compensate participant mining nodes for computations performed. Ethereum provides a decentralized Turing-complete virtual machine, the Ethereum Virtual Machine (EVM), which can execute scripts using an

international network of public nodes. "Gas", an internal transaction pricing mechanism, is used to mitigate spam and allocate resources on the network.

Characteristic

Further information: Cryptocurrency

As with other cryptocurrencies, the validity of each ether is provided by a blockchain, which is a continuously growing list of records, called blocks, which are linked and secured using cryptography.By design, the blockchain is inherently resistant to modification of the data. It is an open, distributed ledger that records transactions between two parties efficiently and in a verifiable and permanent way. Unlike Bitcoin, Ethereum operates using accounts and balances in a manner called state transitions. This does not rely upon unspent transaction outputs (UTXOs). State denotes the current balances of all accounts and extra data. State is not stored on the blockchain, it is stored in a separate Merkle Patricia tree. A cryptocurrency wallet stores the public and private "keys" or "addresses" which can be used to receive or spend Ether. These can be generated through BIP 39 style mnemonics for a BIP 32 "HD Wallet". In Ethereum, this is unnecessary as it does not operate in a UTXO scheme. With the private key, it is possible to write in the blockchain, effectively making an ether transaction.To send ether to an account, you need the public key of that account. Ether accounts are pseudonymous in that they are not linked to individual persons, but rather to one or more specific addresses.[52] Owners can store these addresses in software, on paper and possibly in memory ("brain wallet").

Address :

Ethereum addresses are composed of the prefix "0x", a common identifier for hexadecimal, concatenated with the rightmost 20 bytes of the Keccak-256 hash (big endian) of the ECDSA public key. In hexadecimal, 2 digits represents a byte, meaning addresses contain 40 hexadecimal digits. One example is 0xb794f5ea0ba39494ce839613fffba74279579268, the Poloniex ColdWallet. Contract addresses are in the same format, however they are determined by sender and creation transaction nonce.[53] User accounts are indistinguishable from contract accounts given only an address for each and no blockchain data. Any valid Keccak-256 hash

put into the described format is valid, even if it does not correspond to an account with a private key or a contract. This is unlike Bitcoin, which uses base58check to ensure that addresses are properly typed.

2.3.1.2 Virtual Machine

The Ethereum Virtual Machine (EVM) is the runtime environment for smart contracts in Ethereum. It is a 256-bit register stack, designed to run the same code exactly as intended. It is the fundamental consensus mechanism for Ethereum. The formal definition of the EVM is specified in the Ethereum Yellow Paper. It is sandboxed and also completely isolated from the network, filesystem or other processes of the host computer system. Every Ethereum node in the network runs an EVM implementation and executes the same instructions. In February 1, 2018, there were 27,500 nodes in the main Ethereum network.Ethereum Virtual Machines have been implemented in C++, Go, Haskell, Java, JavaScript, Python, Ruby, Rust, and WebAssembly (currently under development). The Ethereum-flavoured WebAssembly (dubbed "e-WASM") is expected to become a major component of the "Web 3.0", a World Wide Web where users interact with smart contracts through a browser.

2.3.1.3 Ethereum Wallet

Accounts represent identities of external agents (e.g., human personas, mining nodes or automated agents). Accounts use public key cryptography to sign transaction so that the EVM can securely validate the identity of a transaction sender.

Accounts play a central role in Ethereum: they are essential for users to interact with the Ethereum blockchain via transactions.

Account address

Account addresses are the last 20 bytes of a SHA-3 hash of a public key which is generated at account creation. This address is most often in this form : 0x0421d217d2D572244c16e6dc58E87C10d73579UF

- **Mist ethreum wallet**

  The Mist Ethereum wallet, and its parent Mist project, are being developed under the auspices of the Ethereum Foundation. It is the GUI-based option for creating accounts with the geth command.

2.4 CONCEPT USED

- **SOLIDITY**

  Contracts live on the blockchain in a Ethereum-specific binary format called Ethereum Virtual Machine (EVM) bytecode. Contracts are typically written in some high level language such as Solidity and then compiled into bytecode to be uploaded on the blockchain. Solidity is the DEV-created (i.e. Ethereum Foundation-created), Javascript-inspired statically-typed language that can be used to create smart contracts on the Ethereum blockchain. There are other languages you can use as well (LLL, Serpent, etc). The main points in favour of Solidity is that it is statically typed and offers many advanced features like inheritance, libraries, complex user-defined types and a bytecode optimizer.The best way to try out Solidity right now is using the Browser-Based Compiler.

2.5 Dataset

To the best of our knowledge, there is previous knowledge of traditional EMI system works is used as a dataset to identify and remove the ambiguities between the consumer and vendor by developing a good smart contract amongst them.

2.6 Hardware Requirements

1. Hardware Platform: The workstation device should at least have following configurations:
    - 4 GB RAM
    - 1 GB storage
2. Previous Published paper related to topic.

2.7 Software Requirements

1. Go-Ethereum Protocol
2. Mist Ethereum Wallet
3. JetBrains PhpStorm
4. Web3.js

FLOW CHART

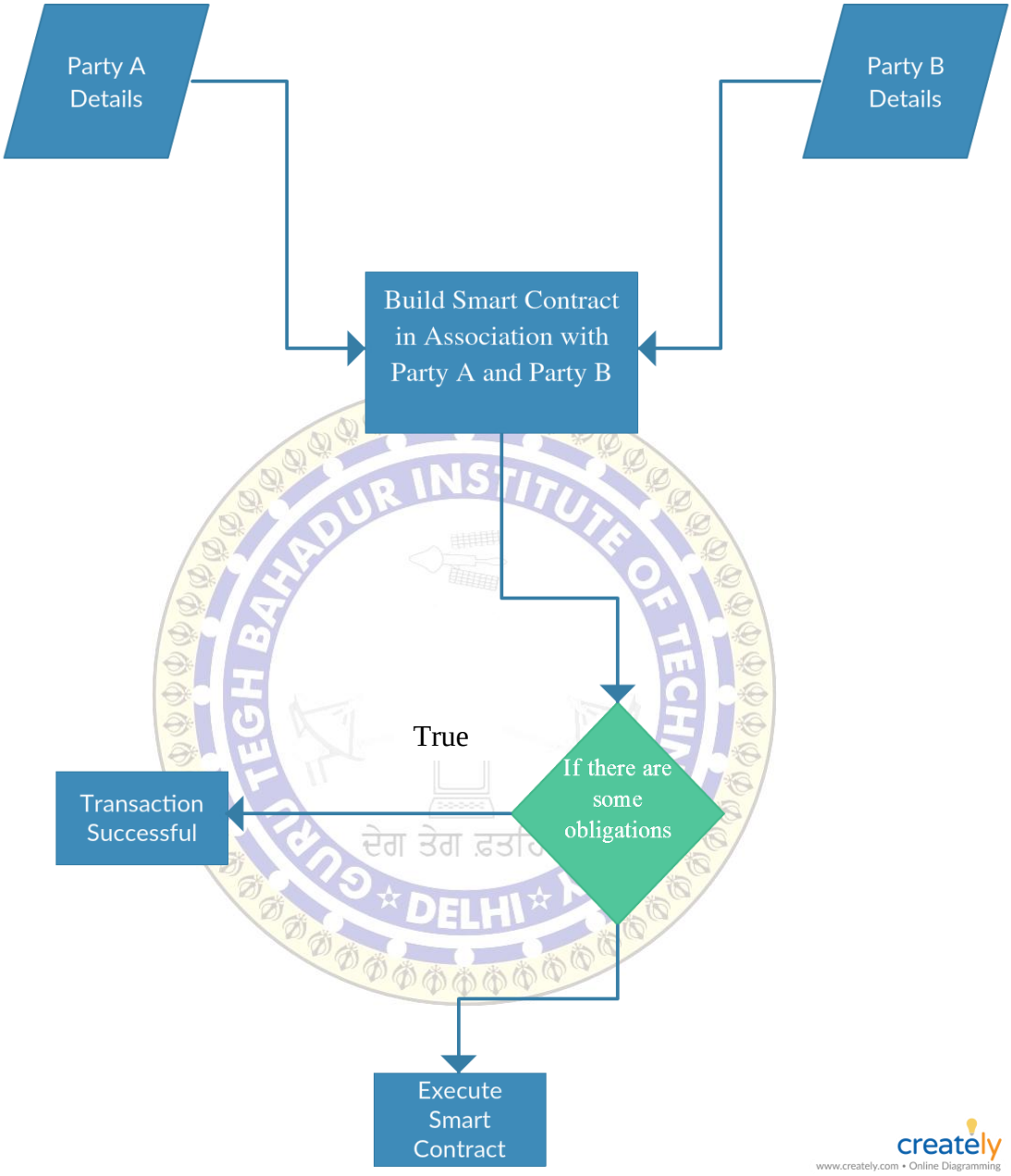


Figure 3.1. Flow of Control for Smart Contract

Data Flow Diagrams (DFDs)

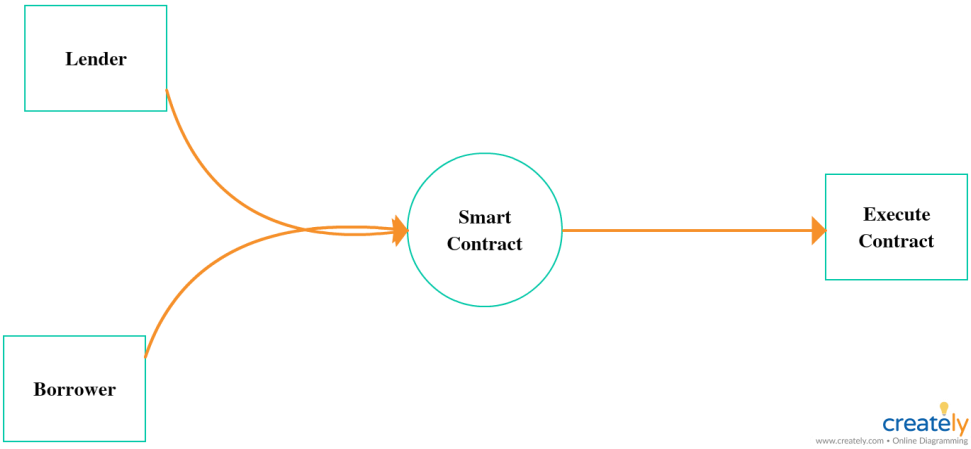


Figure 3.2. Zero Level DFD for EMI Smart Contract

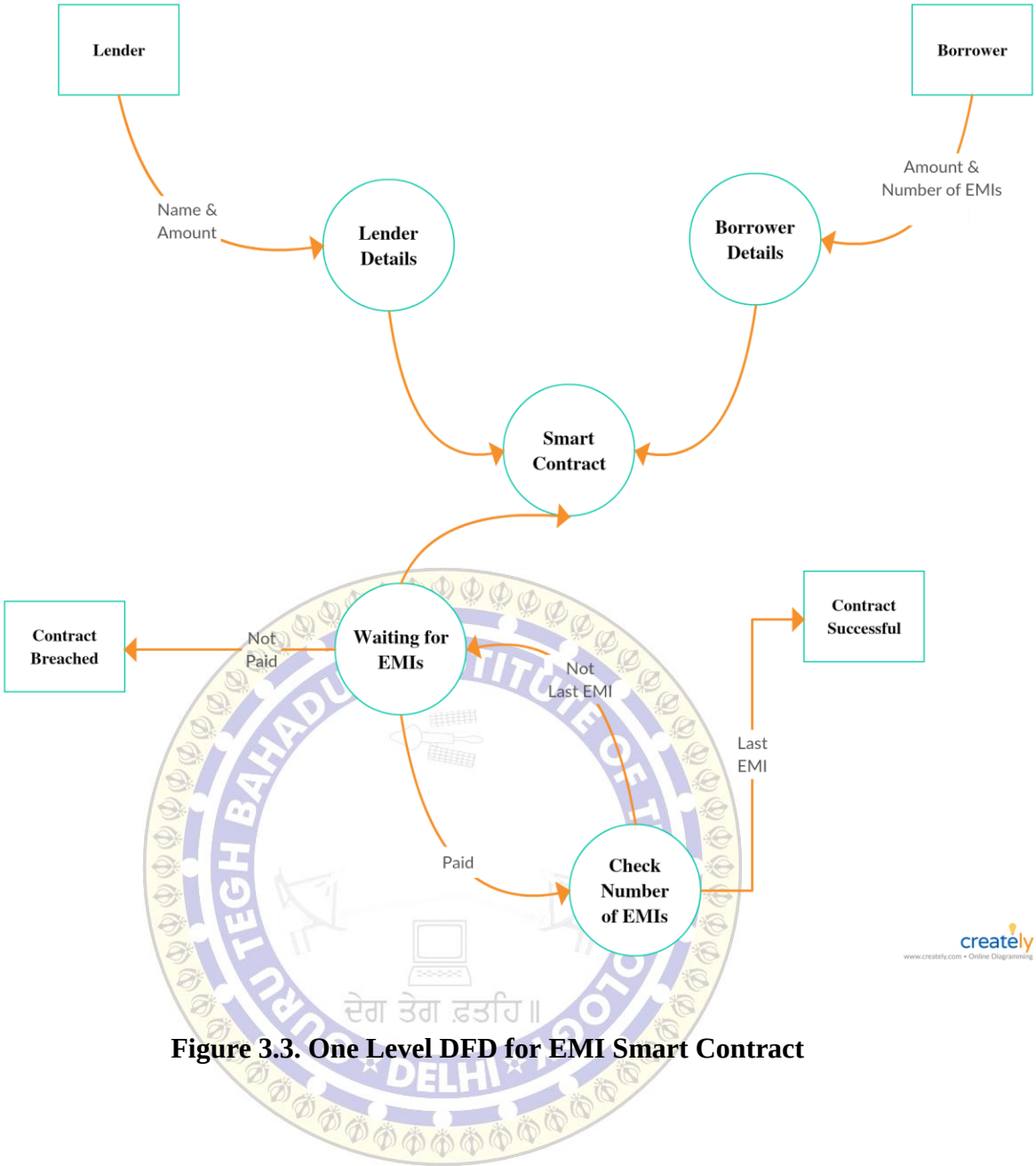


Figure 3.3. One Level DFD for EMI Smart Contract

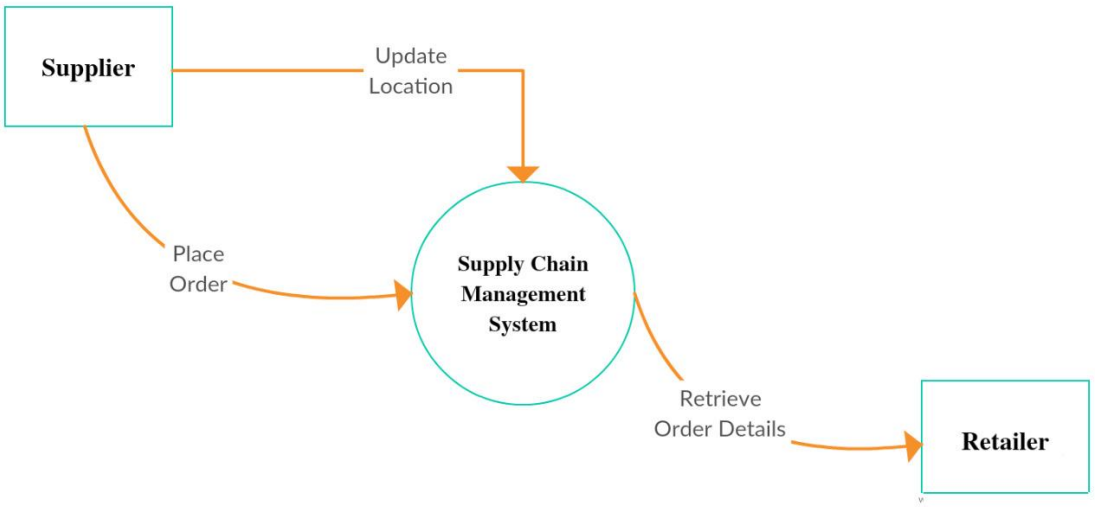


Figure 3.4. Zero Level DFD for SUPPLIER CHAIN MANAGEMENT Smart Contract

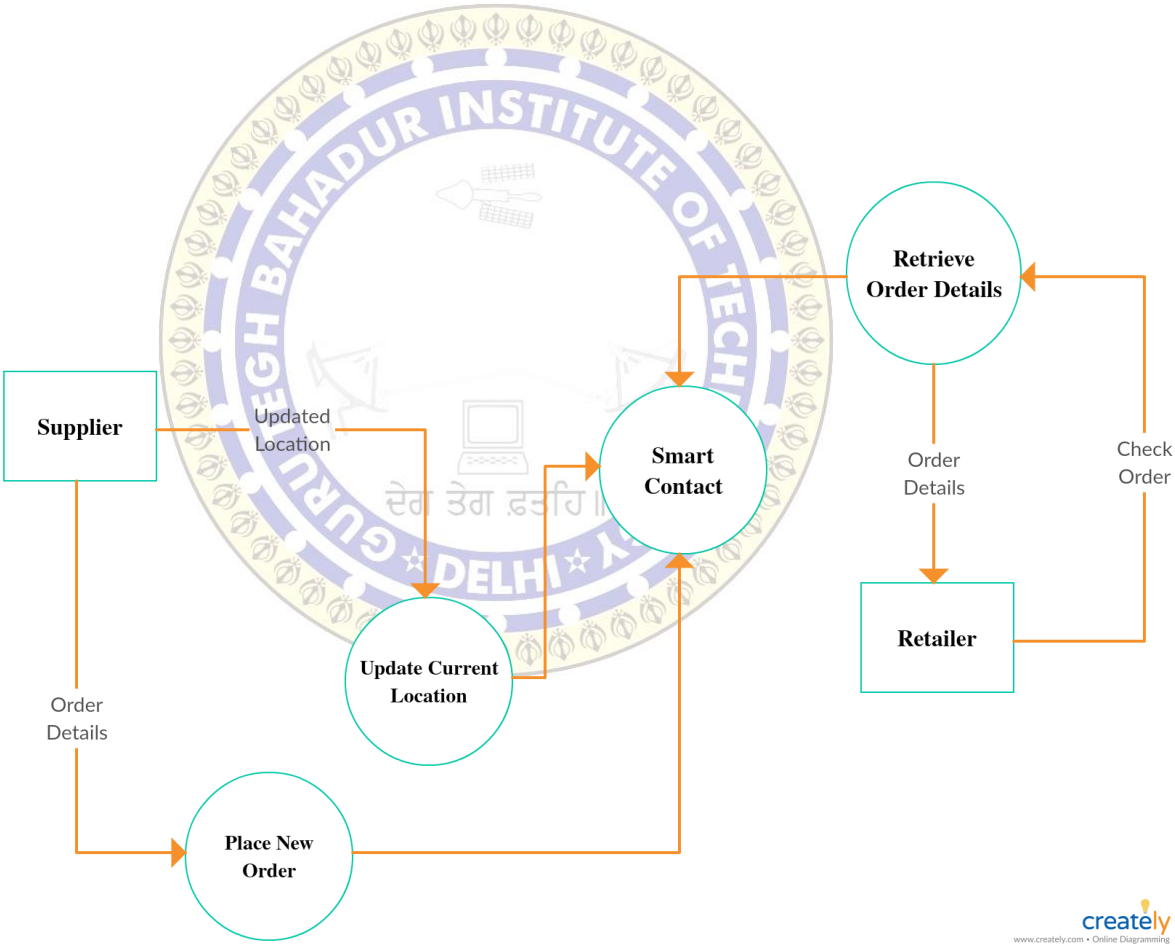


Figure 3.5. One Level DFD for SUPPLIER CHAIN MANAGEMENT Smart Contract

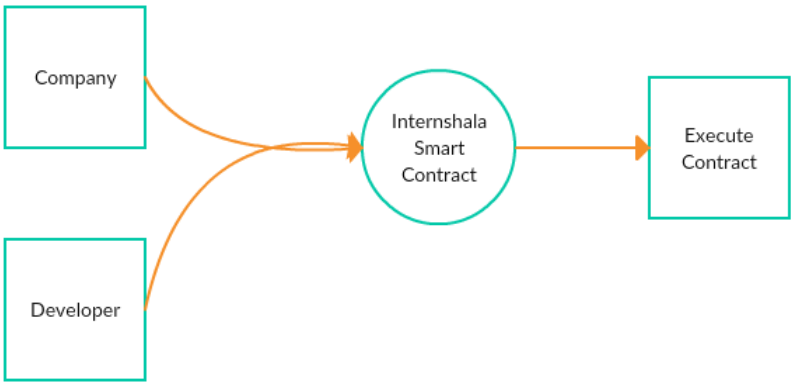


Figure 3.6. Zero Level DFD for Internshala Smart Contract

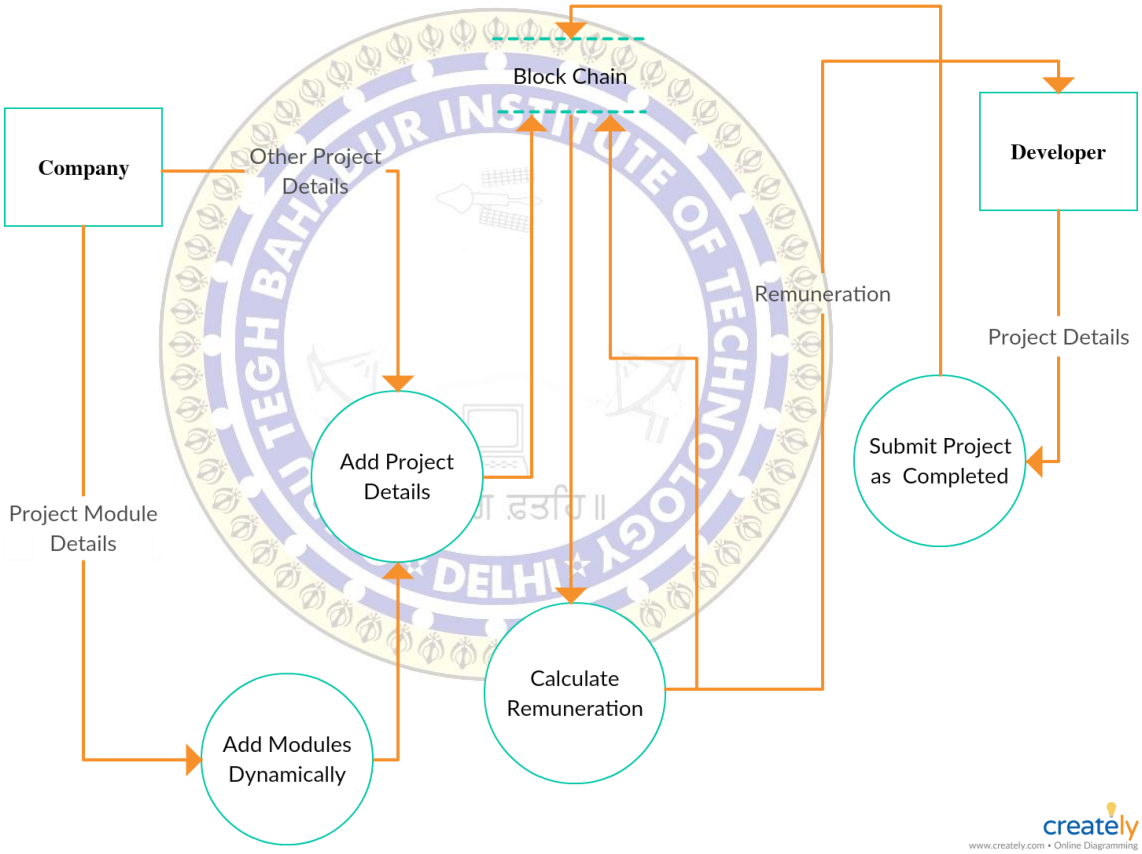


Figure 3.7. One Level DFD for Internshala Smart Contract

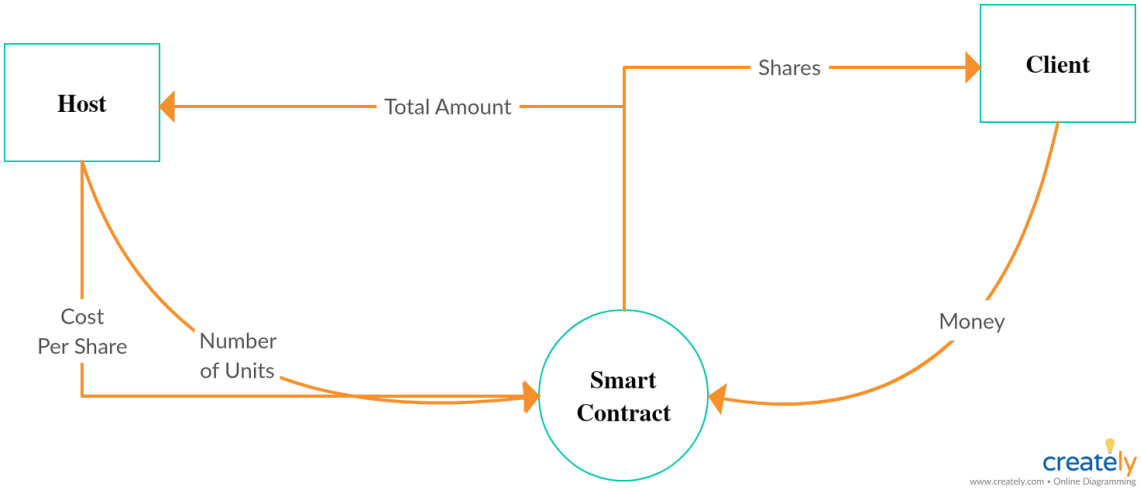


Figure 3.8. Zero Level DFD for Over the Counter Smart Contract

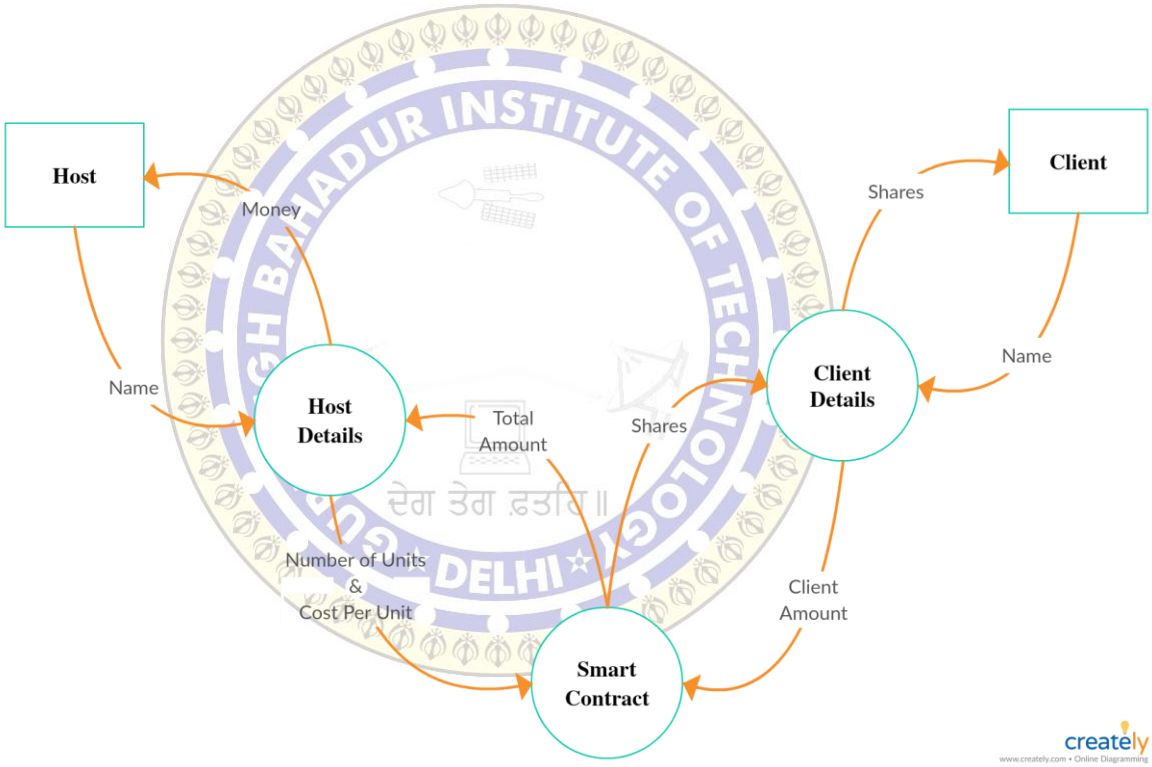


Figure 3.9. One Level DFD for Over the Counter Smart Contract

4.1. Smart Contract

4.1.1. Storage

```
contract SimpleStorage {
uint storedData;
function set(uint x) {
storedData = x;
}
function get() constant returns (uint retVal) {
return storedData;
}
}
```

A contract in the sense of Solidity is a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain. The line uint storedData; declares a state variable called storedData of type uint (unsigned integer of 256 bits). You can think of it as a single slot in a database that can be queried and altered by calling functions of the code that manages the database. In the case of Ethereum, this is always the owning contract. And in this case, the functions set and get can be used to modify or retrieve the value of the variable.

To access a state variable, you do not need the prefix this. as is common in other languages. This contract does not yet do much apart from (due to the infrastructure built by Ethereum) allowing anyone to store a single number that is accessible by anyone in the world without (feasible) a way to prevent you from publishing this number. Of course, anyone could just call set again with a different value and overwrite your number, but the number will still be stored in the history of the blockchain. Later, we will see how you can impose access restrictions so that only you can alter the number.

4.2. Transactions

A blockchain is a globally shared, transactional database. This means that everyone can read entries in the database just by participating in the network. If you want to change something in the database, you have to create a so-called transaction which has to be accepted by all others. The word transaction implies that the change you

want to make (assume you want to change two values at the same time) is either not done at all or completely applied. Furthermore, while your transaction is applied to the database, no other transaction can alter it. As an example, imagine a table that lists the balances of all accounts in an electronic currency. If a transfer from one account to another is requested, the transactional nature of the database ensures that if the amount is subtracted from one account, it is always added to the other account. If due to whatever reason, adding the amount to the target account is not possible, the source account is also not modified. Furthermore, a transaction is always cryptographically signed by the sender (creator). This makes it straightforward to guard access to specific modifications of the database. In the example of the electronic currency, a simple check ensures that only the person holding the keys to the account can transfer money from it.

4.3. The Ethereum Virtual Machine

4.3.1. Overview

The Ethereum Virtual Machine or EVM is the runtime environment for smart contracts in Ethereum. It is not only sandboxed but actually completely isolated, which means that code running inside the EVM has no access to network, filesystem or other processes. Smart contracts even have limited access to other smart contracts.

4.3.2. Accounts

There are two kinds of accounts in Ethereum which share the same address space: External accounts that are controlled by public-private key pairs (i.e. humans) and contract accounts which are controlled by the code stored together with the account.

The address of an external account is determined from the public key while the address of a contract is determined at the time the contract is created (it is derived from the creator address and the number of transactions sent from that address, the so-called “nonce”).

Apart from the fact whether an account stores code or not, the EVM treats the two types equally, though.

Every account has a persistent key-value store mapping 256 bit words to 256 bit words called storage.

Furthermore, every account has a balance in Ether (in “Wei” to be exact) which can be modified by sending transactions that include Ether.

4.3.3. Transactions

A transaction is a message that is sent from one account to another account (which might be the same or the special zero-account, see below). It can include binary data (its payload) and Ether.

If the target account contains code, that code is executed and the payload is provided as input data.

If the target account is the zero-account (the account with the address 0), the transaction creates a new contract. As already mentioned, the address of that contract is not the zero address but an address derived from the sender and its number of transaction sent (the “nonce”). The payload of such a contract creation transaction is taken to be EVM bytecode and executed. The output of this execution is permanently stored as the code of the contract. This means that in order to create a contract, you do not send the actual code of the contract, but in fact code that returns that code.

4.3.4. Gas

Upon creation, each transaction is charged with a certain amount of gas, whose purpose is to limit the amount of work that is needed to execute the transaction and to pay for this execution. While the EVM executes the transaction, the gas is gradually depleted according to specific rules.

The gas price is a value set by the creator of the transaction, who has to pay gas_price * gas up front from the sending account. If some gas is left after the execution, it is refunded in the same way.

If the gas is used up at any point (i.e. it is negative), an out-of-gas exception is triggered, which reverts all modifications made to the state in the current call frame.

4.3.5. Storage, Memory and the Stack

Each account has a persistent memory area which is called storage. Storage is a key-value store that maps 256 bit words to 256 bit words. It is not possible to enumerate storage from within a contract and it is comparatively costly to read and even more so, to modify storage. A contract can neither read nor write to any storage apart from its own.

The second memory area is called memory, of which a contract obtains a freshly cleared instance for each message call. Memory can be addressed at byte level, but read and written to in 32 byte (256 bit) chunks. Memory is more costly the larger it grows (it scales quadratically).

The EVM is not a register machine but a stack machine, so all computations are performed on an area called the stack. It has a maximum size of 1024 elements and contains words of 256 bits. Access to the stack is limited to the top end in the following way: It is possible to copy one of the topmost 16 elements to the top of the stack or swap the topmost element with one of the 16 elements below it. All other operations take the topmost two (or one, or more, depending on the operation) elements from the stack and push the result onto the stack. Of course it is possible to move stack elements to storage or memory, but it is not possible to just access arbitrary elements deeper in the stack without first removing the top of the stack.

4.4 Instruction Set

The instruction set of the EVM is kept minimal in order to avoid incorrect implementations which could cause consensus problems. All instructions operate on the basic data type, 256 bit words. The usual arithmetic, bit, logical and comparison operations are present. Conditional and unconditional jumps are possible. Furthermore, contracts can access relevant properties of the current block like its number and timestamp.

4.4.1. Message Calls

Contracts can call other contracts or send Ether to non-contract accounts by the means of message calls. Message calls are similar to transactions, in that they have a source, a target, data payload, Ether, gas and return data. In fact, every transaction consists of a top-level message call which in turn can create further message calls.

A contract can decide how much of its remaining gas should be sent with the inner message call and how much it wants to retain. If an out-of-gas exception happens in the inner call (or any other exception), this will be signalled by an error value put onto the stack. In this case, only the gas sent together with the call is used up. In Solidity, the calling contract causes a manual exception by default in such situations, so that exceptions "bubble up" the call stack.

As already said, the called contract (which can be the same as the caller) will receive a freshly cleared instance of memory and has access to the call payload - which will be provided in a separate area called the calldata. After it finished execution, it can return data which will be stored at a location in the caller's memory preallocated by the caller.

Calls are limited to a depth of 1024, which means that for more complex operations, loops should be preferred over recursive calls.

4.4.2. Delegatecall / Callcode and Libraries

There exists a special variant of a message call, named delegate call which is identical to a message call apart from the fact that the code at the target address is executed in the context of the calling contract and msg.sender and msg.value do not change their values.

This means that a contract can dynamically load code from a different address at runtime. Storage, current address and balance still refer to the calling contract, only the code is taken from the called address.

This makes it possible to implement the "library" feature in Solidity: Reusable library code that can be applied to a contract's storage in order to e.g. implement a complex data structure.

4.4.3. Logs

It is possible to store data in a specially indexed data structure that maps all they way up to the block level. This feature called logs is used by Solidity in order to implement events. Contracts cannot access log data after it has been created, but they can be efficiently accessed from outside the blockchain. Since some part of the log data is stored in bloom filters, it is possible to search for this data in an efficient and cryptographically secure way, so network peers that do not download the whole blockchain (“light clients”) can still find these logs.'

4.4.4. Create

Contracts can even create other contracts using a special opcode (i.e. they do not simply call the zero address). The only difference between these create calls and normal message calls is that the payload data is executed and the result stored as code and the caller / creator receives the address of the new contract on the stack.

4.4.5. Selfdestruct

The only possibility that code is removed from the blockchain is when a contract at that address performs the SELFDESTRUCT operation. The remaining Ether stored at that address is sent to a designated target and then the storage and code is removed.
Note that even if a contract’s code does not contain the SELFDESTRUCT opcode, it can still perform that operation using delegatecall or callcode.

Result

There will be an web application that can execute a contract between a vendor and the consumer to enhance the execution of obligations between them through smart contract. This contract is autonomous, secure and decentralized.

Conclusion and Future Work

At the final stage of our major project, the application takes some parameters from the user like user's name, contact number, total amount and number of emi.

After that emi amount is calculated and then a smart contract is build in authorization with vendor and consumer. After the contract is build then if there is any obligation from vendor or consumer side then contract get executed to bear the reasonable charges.

Extension of Project:

- Usage of Smart Contract in Health Sector.
- Usage of Smart Contract in E-Trading.

REFERENCES

[1] Solidity Documentation - https://solidity.readthedocs.io/en/v0.4.20/

[2] Go Ethereum Protocol - https://github.com/ethereum/go-ethereum

[3] Mist Wallet - https://github.com/ethereum/mist/

[4] Udemy Blockchain Developer - https://www.udemy.com/blockchain-developer/

[5] Udemy Ethereum Masterclass - https://www.udemy.com/ethereum-masterclass/

[6] https://www.coindesk.com/information/ethereum-smart-contracts-work/

[7] https://medium.com/crypto-currently/build-your-first-smart-contract-fc36a8ff50ca

[8] http://farzicoder.com/Setting-up-Geth-Mist-TestRPC-Part-1-Getting-Start-with-Smart-Contracts/

[9] https://www-01.ibm.com/software/commerce/offers/pdfs/Blockchain_3-15-2017.pdf

[10] More Legal Aspects of Smart Contract Applications - https://www.virtualcurrencyreport.com/wp-content/uploads/sites/13/2018/03/Perkins-Coie-LLP-More-Legal-Aspects-of-Smart-Contract-Applications-White-Paper.pdf

[11] http://www.itrelease.com/2017/11/difference-centralised-decentralised-distributed-processing/

[12] https://www.weforum.ord/agenda/2018/03/blockchain-bitcoin-explainer-shiller-roubini/

[13] http://gavwood.com/paper.pdf

[14] Bitcoin or Ethereum - The Million Dollar Question ? https://www.economist.com/sites/default/files/carey_business_school_submission.pdf

[15] Ethereum - http://fps2017.loria.fr/wp-content/uploads/2017/10/19.pdf

[16] https://github.com/ethereum/wiki/wiki/White-Paper

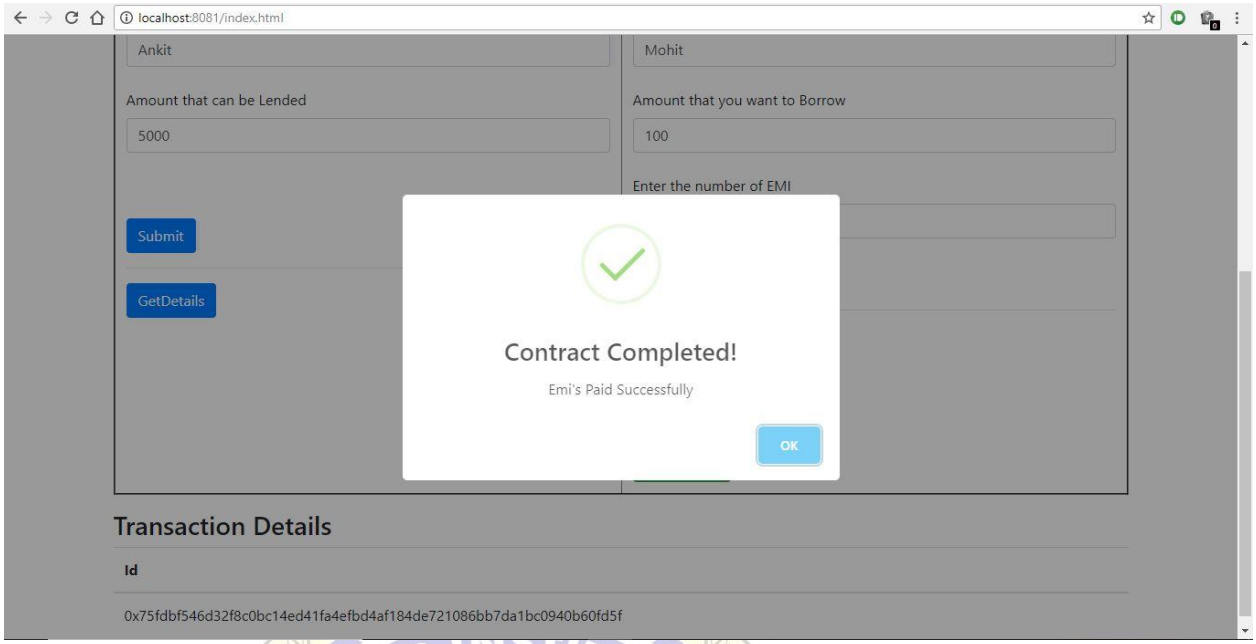
[17] What is Smart Contract ? A Beginner's Guide - https://www.youtube.com/watch?v=qdoUpGg_DpQ

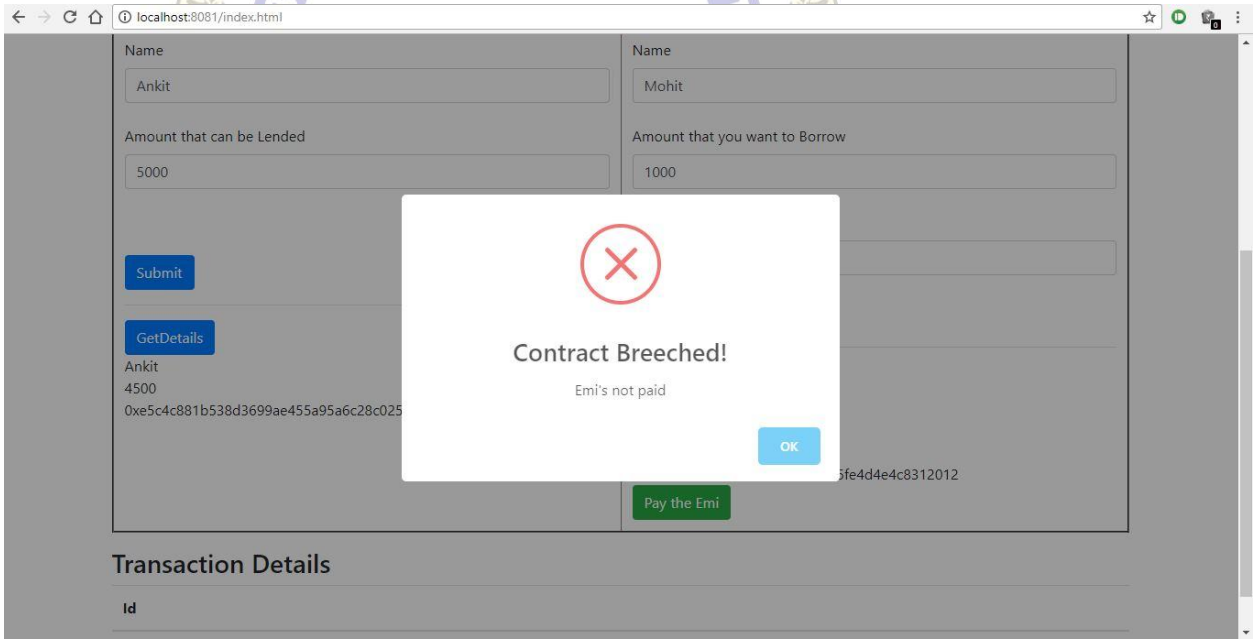
[18] A Guide to Building Your First Decentralized Application - https://www.youtube.com/watch?v=gSQXq2_j-mw&feature=youtu.be

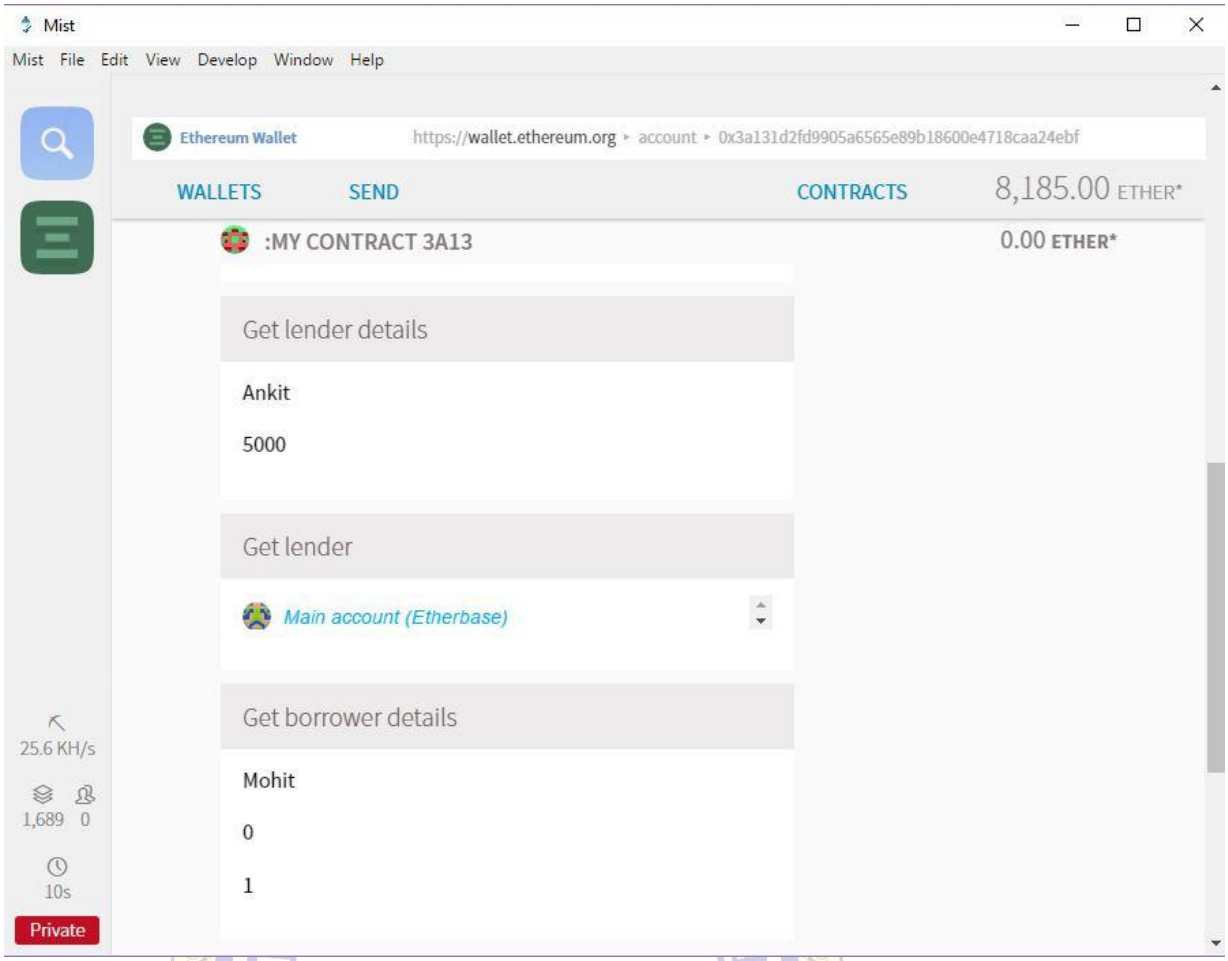
[19] Beginner's Guide : Smart Contracts Programming Tutorial in Solidity 3 - https://www.youtube.com/watch?v=lDa8AOH51_0&feature=youtu.be

[20] Developing Ethereum Smart Conracts - https://www.youtube.com/watch?v=KU6bvciWgRE&list=PL0lNJEnwfVVMuX2Ds19Wj_7Mcze3FDJr3

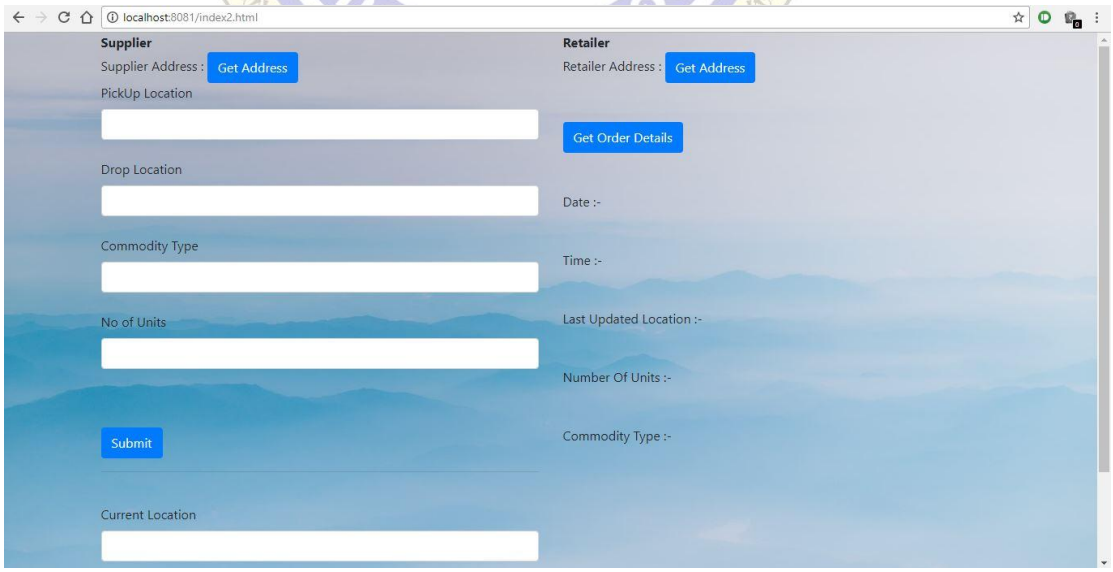
EMI CONTRACT OUTPUT :

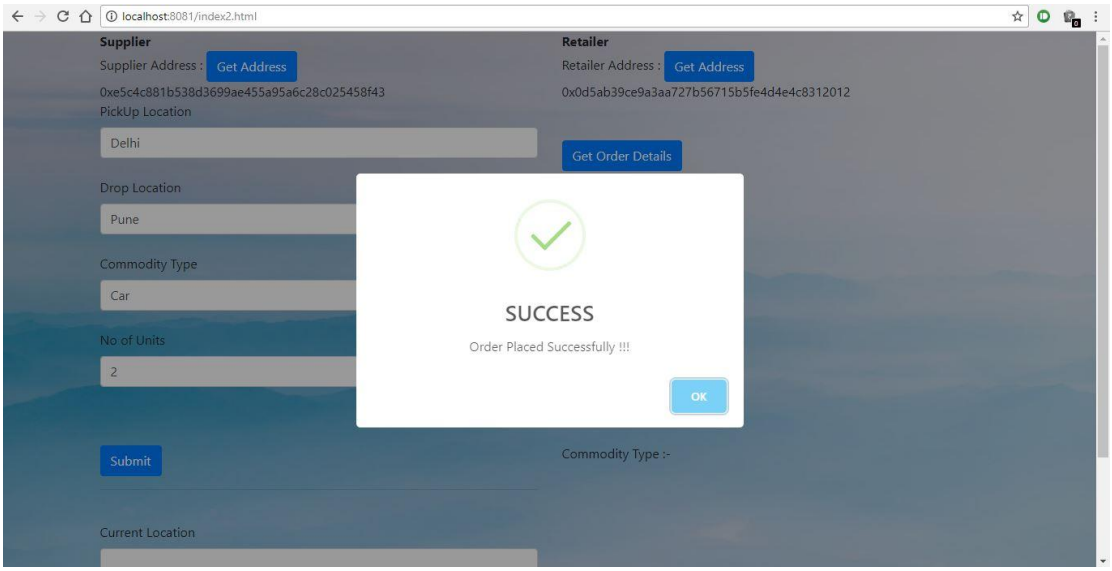


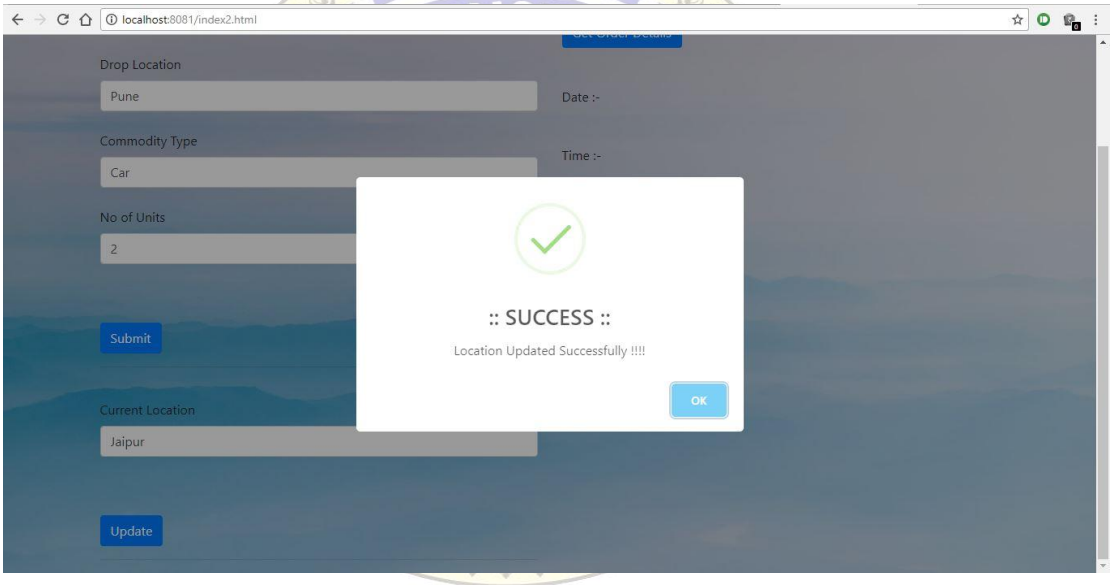


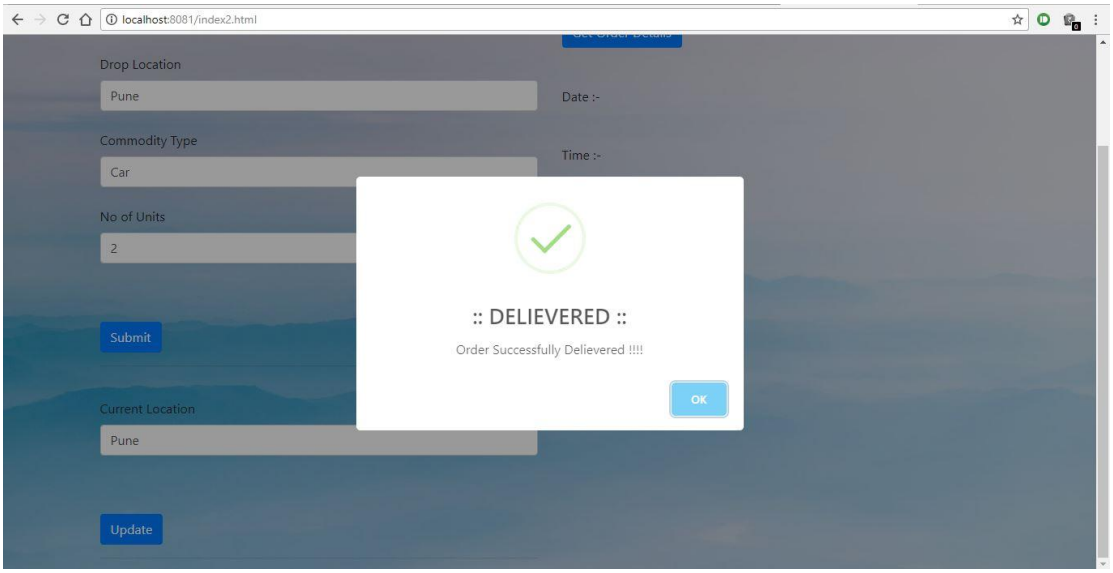


DELIVERY CONTRACT OUTPUT :




localhost:8081/index2.html
Supplier
Supplier Address :
Get Address
0xe5c4c881b538d3699ae455a95a6c28c025458f43
PickUp Location
Delhi
Drop Location
Pune
Commodity Type
Car
No of Units
2
Submit
Current Location
Retailer
Retailer Address :
Get Address
0x0d5ab39ce9a3aa727b56715b5fe4d4e4c8312012
Get Order Details
Commodity Type :-
SUCCESS
Order Placed Successfully !!!
OK


localhost:8081/index2.html
Drop Location
Pune
Commodity Type
Car
No of Units
2
Submit
Current Location
Jaipur
Update
Date :-
Time :-
:: SUCCESS ::
Location Updated Successfully !!!!
OK



INDEX.HTML (EMI)

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>EMI</title>
   <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6J
Xm" crossorigin="anonymous">
</head>
<style type="text/css">
   table
   {
      table-layout: fixed;
   }
</style>
<script src="web3.js/dist/web3.min.js"></script>
<script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>
<script type="text/javascript">



   var contract_address = "0x3a131D2fd9905a6565E89B18600e4718Caa24EbF";
   var contract_abi =[ { "constant": true, "inputs": [], "name": "getBorrower",
"outputs": [ { "name": "", "type": "address", "value":
"0x0000000000000000000000000000000000000000" } ], "payable": false,
"stateMutability": "view", "type": "function" }, { "constant": false, "inputs": [ {
"name": "name", "type": "string" }, { "name": "value", "type": "uint256" } ], "name":
"setLenderDetails", "outputs": [], "payable": false, "stateMutability": "nonpayable",
"type": "function" }, { "constant": false, "inputs": [ { "name": "t", "type": "address" }
], "name": "setLender", "outputs": [], "payable": false, "stateMutability":
"nonpayable", "type": "function" }, { "constant": true, "inputs": [], "name": "getEmi",
"outputs": [ { "name": "", "type": "uint256", "value": "0" } ], "payable": false,
"stateMutability": "view", "type": "function" }, { "constant": false, "inputs": [],
"name": "transfer", "outputs": [], "payable": false, "stateMutability": "nonpayable",
"type": "function" }, { "constant": true, "inputs": [], "name": "getLenderDetails",
"outputs": [ { "name": "", "type": "string", "value": "" }, { "name": "", "type":
"uint256", "value": "0" } ], "payable": false, "stateMutability": "view", "type":
"function" }, { "constant": true, "inputs": [], "name": "getLender", "outputs": [ {
"name": "", "type": "address", "value":
"0x0000000000000000000000000000000000000000" } ], "payable": false,
"stateMutability": "view", "type": "function" }, { "constant": true, "inputs": [],
"name": "getBorrowerDetails", "outputs": [ { "name": "", "type": "string", "value": ""
}, { "name": "", "type": "uint256", "value": "0" }, { "name": "", "type": "uint256",
```

```
"value": "0" } ], "payable": false, "stateMutability": "view", "type": "function" }, { "constant": false, "inputs": [ { "name": "t", "type": "address" } ], "name": "setBorrower", "outputs": [], "payable": false, "stateMutability": "nonpayable", "type": "function" }, { "constant": false, "inputs": [ { "name": "name", "type": "string" }, { "name": "value", "type": "uint256" }, { "name": "noofemi", "type": "uint256" } ], "name": "setBorrowerDetails", "outputs": [], "payable": false, "stateMutability": "nonpayable", "type": "function" }, { "payable": true, "stateMutability": "payable", "type": "fallback" } ];

   if (typeof web3 !== 'undefined') {
      web3 = new Web3(web3.currentProvider);
   } else {
      // set the provider you want from Web3.providers
      web3 = new Web3(new Web3.providers.HttpProvider("http://127.0.0.1:8545"));
   }

   var contract_instance = web3.eth.contract(contract_abi).at(contract_address);
   var s3 = contract_instance.getBorrowerDetails();
   var boolArray = new Boolean(s3[2]+1);
   var b1 = new Boolean(false);

   for (var b in boolArray) b = false;

   var i = 0;

   function lenderProcess() {

      var s1 = document.getElementById("name").value.toString();
      var s2 = document.getElementById("amount").value;
      web3.personal.unlockAccount(web3.eth.accounts[0],"test1234");

contract_instance.setLenderDetails(s1,parseInt(s2),{from:web3.eth.accounts[0],gas:2
00000},function (error,result)
      {
         if(error)
         {
            console.error(error);
         }
         else
         {
            var txHash = result;
            console.log(txHash);
            document.getElementById("Tran").innerText = txHash.toString();
         }
      });
   }

   function borrowerProcess() {
```

```
        var s1 = document.getElementById("name1").value.toString();
        var s2 = document.getElementById("amount1").value;
        var s3 = document.getElementById("emi").value;
        web3.personal.unlockAccount(web3.eth.accounts[1],"test1234");

contract_instance.setBorrowerDetails(s1,parseInt(s2),parseInt(s3),{from:web3.eth.acc
ounts[1],gas:200000},function (error,result)
        {
           if(error)
           {
              console.error(error);
           }
           else
           {
              var txHash = result;
              console.log(txHash);
              document.getElementById("Tran").innerText = txHash.toString();
           }
        });
        boolArray[i]=true;
        checker();
           }

    function checker() {

        if(boolArray[i]==true)
        {
           i++;
           console.log("in Function Checker in If ");
           setTimeout(checker,3000);
        }
        else
        {
           console.log("in Function Checker in else");
           if(b1==true)
           {
              swal("Contract Breeched!", "Emi's not paid", "error");
           }
           else {
              b1 = true;
              setTimeout(checker, 30000);
           }
        }
    }

    <!-- setTimeout(checker,3000);-->
```

```
   function getLenderDeatils()
   {
      var s = contract_instance.getLenderDetails();
      document.getElementById("GetLenDetName").innerText =s[0];
      document.getElementById("GetLenDetAmt").innerText =s[1];
      document.getElementById("GetLender").innerText =
contract_instance.getLender().toString();

   }
   function getBorrowerDeatils()
   {

      var s = contract_instance.getBorrowerDetails();
      document.getElementById("GetBowDetName").innerText = s[0];
      document.getElementById("GetBowDetAmt").innerText = s[1];
      document.getElementById("GetBowDetNoe").innerText = s[2];
      document.getElementById("GetBorrower").innerText =
contract_instance.getBorrower().toString();
   }

   function getLenderAddress() {
      document.getElementById("getLen").innerText =
(web3.eth.accounts[0]).toString();
      web3.personal.unlockAccount(web3.eth.accounts[0],"test1234");

contract_instance.setLender(web3.eth.accounts[0],{from:web3.eth.accounts[0],gas:20
0000},function (error,result)
      {
         if(error)
         {
            console.error(error);
         }
         else
         {
            var txHash = result;
            console.log(txHash);
            document.getElementById("Tran").innerText = txHash.toString();
         }
      });
   }
   function getBorrowerAddress() {
      document.getElementById("getBow").innerText =
(web3.eth.accounts[1]).toString();
      web3.personal.unlockAccount(web3.eth.accounts[1],"test1234");

contract_instance.setBorrower(web3.eth.accounts[1],{from:web3.eth.accounts[1],gas:
220000},function (error,result)
      {
```

```
            if(error)
            {
               console.error(error);
            }
            else
            {
               var txHash = result;
               console.log(txHash);
               document.getElementById("Tran").innerText = txHash.toString();
            }
         });

      }
      function transfer() {

         boolArray[i++]=true;
         web3.personal.unlockAccount(web3.eth.accounts[0],"test1234");
         contract_instance.transfer({from:web3.eth.accounts[0],gas:300000},function
(error,result)
         {
            if(error)
            {
               console.error(error);
            }
            else
            {
               var txHash = result;
               console.log(txHash);
               document.getElementById("Tran").innerText = txHash.toString();
            }
         });
         var s = contract_instance.getBorrowerDetails();
         if(s[1]==0)
         {
            swal("Contract Completed!", "Emi's Paid Successfully", "success");
         }
      }

</script>
<body class="container">
<h1 class="jumbotron">EMI Contract</h1>
<div class="table-responsive">
   <table class="table" border="2" width="100%">
      <th> Party A <hr><br> Lender Address : <button class="btn btn-primary"
onclick="getLenderAddress()">Get Address</button>
         <span id="getLen"></span>
      </th>
```

```
<th> Party B <hr><br> Borrower Address : <button class="btn btn-primary" onclick="getBorrowerAddress()">Get Address</button>
  <span id="getBow"></span>
</th>
<tr>
  <td>
    <form action="javascript:lenderProcess()">
      <label>Name</label>
      <input type="text" class="form-control" id="name" /><br>
      <label>Amount that can be Lended</label>
      <input type="number" class="form-control" id="amount" /><br>
      <br><br>
      <input type="submit" class="btn btn-primary" value="Submit"/>
      <hr>
    </form>
    <button class="btn btn-primary" onclick="getLenderDeatils()">GetDetails</button>
    <br>
    <span id ="GetLenDetName"></span><br>
    <span id ="GetLenDetAmt"></span><br>
    <span id="GetLender"></span>
  </td>
  <td>
    <form action="javascript:borrowerProcess()">
      <label>Name</label>
      <input type="text" id="name1" class="form-control"/><br>

      <label>Amount that you want to Borrow </label>
      <input type="number" id="amount1" class="form-control" /><br>

      <label>Enter the number of EMI</label>
      <input type="number" class="form-control" id="emi" /><br>

      <input type="submit" class="btn btn-primary" value="Submit"/>
      <hr>

    </form>
    <button class="btn btn-primary" onclick="getBorrowerDeatils()">GetBorrower</button>
    <br>
    <span id ="GetBowDetName"></span><br>
    <span id ="GetBowDetAmt"></span><br>
    <span id ="GetBowDetNoe"></span><br>
    <span id ="GetBorrower"></span>
    <br>
    <button class="btn btn-success" onclick="transfer()">Pay the Emi</button>
  </td>
```

```
        </tr>
    </table>
</div>

<h3>Transaction Details</h3>
<div class="table-responsive">
    <table class="table">
        <thead>
        <tr>
            <th>Id</th>
        </tr>
        </thead>
        <tbody>
        <tr>
            <td id="Tran"></td>
        </tr>
        </tbody>

    </table>
</div>


<script        src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl
" crossorigin="anonymous"></script>
</body>
</html>
```

SMART CONTRACT (EMI)

```
pragma solidity ^0.4.8;

contract myContract{
   address lender;
   address borrower;
   uint lamt;
   string bname;
   uint bamt;
   string lname;
   uint emi;
   uint noe;
   uint i;

   function setLenderDetails(string name,uint value) public
   {
    lname = name;
    lamt = value;
   }

   function setBorrowerDetails(string name,uint value,uint noofemi) public
   {
    bname = name;
      require(value<=lamt);
    bamt = value;
    noe = noofemi;
    emi = bamt/noofemi;
    lamt-=bamt;
    i=1;
   }


   function setLender(address t) public {
      lender =t;
   }

   function setBorrower(address t) public {
      borrower =t;
   }

   function getLender() public constant returns(address){
      return lender;
   }

   function getBorrower() public constant returns(address){
      return borrower;
   }
```

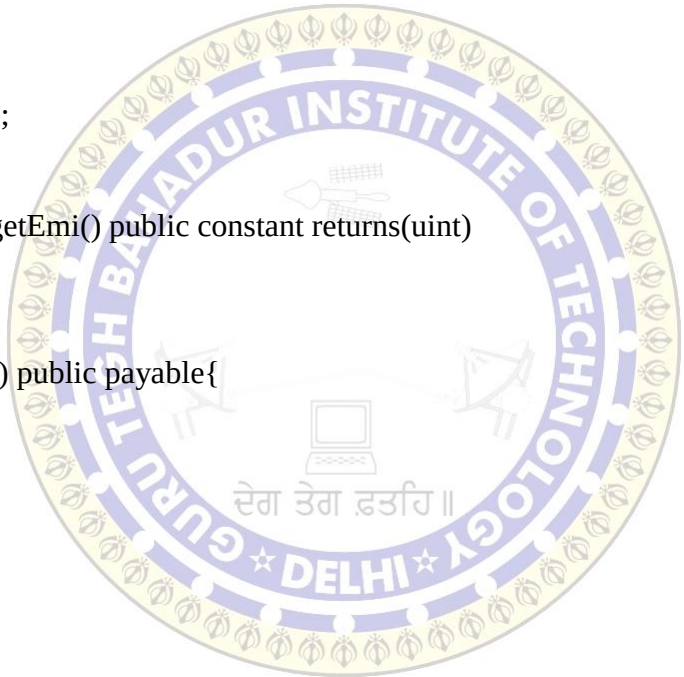
```
    function getLenderDetails() public constant returns(string,uint)
    {
     return (lname,lamt);
    }

    function getBorrowerDetails() public constant returns(string,uint,uint)
    {
     return (bname,bamt,noe);
    }

    function transfer() public
    {
       if(i<=noe)
       {
       lamt+=emi;
       bamt-=emi;
       i++;
       }
       else
       revert();
    }

    function getEmi() public constant returns(uint)
    {
    return i;
    }
    function () public payable{

    }
}
```



SMART CONTRACT (DELIVERY)

```
pragma solidity ^0.4.8;
contract myContract{
   address supplier_id;
   address retailer_id;
   string pickup_loc;
   string date;
   string time;
   string curr_loc;
   string drop_loc;
   string comm_type;
   uint no_of_units;
   function setSupplier(address t)public{
      supplier_id=t;
   }
   function setRetailer(address t)public{
      retailer_id=t;
   }
   function placeOrder(string pick,string drop,string comm,uint nou,string
curr_time,string curr_date) public
   {
      date=curr_date;
      time=curr_time;
      pickup_loc=pick;
      drop_loc=drop;
      comm_type=comm;
      no_of_units=nou;
      curr_loc=pick;

   }
   function checkDelievery() public constant returns(bool)
   {
      if(keccak256(drop_loc)==keccak256(curr_loc))
      return true;
      else
      return false;
   }
   function updateOrder(string curr_date,string curr_time,string loc) public
   {
      date=curr_date;
      time=curr_time;
      curr_loc=loc;
   }
   function returnSupplier() public constant returns(address){
      return supplier_id;
   }
   function returnRetailer() public constant returns(address){
```

```
        return retailer_id;
    }
    function returnDropLocation() public constant returns(string)
    {
        return drop_loc;
    }
    function orderDetails() public constant returns(string,string,string,string,uint){
      return (date,time,curr_loc,comm_type,no_of_units);
    }


    function () public payable{

    }

}
```