**Decentralized Applications**

*(Using Ethereum Blockchain)*

*Project Submitted in Partial Fulfilment of Requirement*

*for the Award of the Degree of*

*Bachelor of Technology*

*In*

**COMPUTER SCIENCE AND ENGINEERING**

*By*

**PRAMESH BAJRACHARYA ............ 14221A0566**

**ABHISHEK SHRESTHA ................... 14221A0563**

**SAURAV BAJRACHARYA ............... 14221A0567**

**SUNDAR GHIMIRE ........................... 14221A0576**

*Under The Esteemed Guidance of*

***Mr. N.KRISHNAIAH***

**B.Tech.,M.Tech.,(Ph.D), MISTE**

Associate Professor, CSE Dept

**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**BONAM VENKATA CHALAMAYYA ENGINEERING COLLEGE**

Accredited by Approved by N.B.A, Approved by A.I.C.T.E, New Delhi & Affiliated to J.N.T.U, Kakinada

Odalarevu-533210, Amalapuram, E.G.Dist.

2014-2018

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**BONAM VENKATA CHALAMAYYA ENGINEERING COLLEGE**

Accredited by Approved by N.B.A, Approved by A.I.C.T.E, New Delhi & Affiliated to J.N.T.U, Kakinada

Odalarevu-533210, Amalapuram, E.G.Dist.



***CERTIFICATE***

This is to certify that this project work titled **“DECENTRALIZED APPLICATIONS *(Using ETHEREUM BLOCKCHAIN)*”** is a bonafide work carried out by the following students under students under our guidance and supervision.

**Name of the Student Regd no:**

**PRAMESH BAJRACHARYA ............ 14221A0566**

**ABHISHEK SHRESTHA ................... 14221A0563**

**SAURAV BAJRACHARYA ............... 14221A0567**

**SUNDAR GHIMIRE ........................... 14221A0576**

We do by accord our approval of it as project work carried out and presented in a manner required for its acceptance fulfilment for the award of the **Degree of Bachelor of Technology in COMPUTER SCIENCE AND ENGINEERING** by **Jawaharlal Nehru Technology University, Kakinada** during the year 2017-2018. The result embodied in the project has not been submitted to any other universities or institute for the award of any degree or diploma.

PROJECT GUIDE: HEAD OF DEPARTMENT:

**Mr. N Krishnaiah Mr. Dr G Jena**

**B.Tech., M.Tech.,(Ph.D), M.E,Ph.D,** Associate Professor, Professor, Department of C.S.E Department of C.S.E

# ACKNOWLEDGEMENT

We would like take this opportunity to express our heartiest concern to words all those people who have helped us, in various ways to complete this project

We thank **Mr. N.Krishnaiah,** project guide, for his insistence on good programming techniques which helped us to design and develop a successful model of **“DECENTRALIZED APPLICATIONS *(Using ETHEREUM BLOCKCHAIN)*”.**

Weexpress thanks and graduated to **Mr.Dr G Jena,** Head of the department Computer Science and Engineering B.V.C Engineering College for his encouraging support and guidance in carrying out the project.

Express our gratitude to **Dr. D.S.V.Prasad**, Principle of our Institute for forecasting an excellent academic environment which made our project work possible.

We are very grateful to the faculty members of the Department of Computer Science And Engineering, B.V.C Engineering College, Odalarevu for their help in completion of project.

**Project Associates**

Pramesh Bajracharya,

Abhishek Shrestha,

Saurav Bajracharya,

Sundar Ghimire.

# ABSTRACT

**Decentralized applications (dApps)** are applications that run on a P2P network of computers rather than a single computer. A new model for building successful and massively scalable applications is emerging. Bitcoin led the way with its open-source, peer-to-peer nature, cryptographically-stored records (block chain), and limited number of tokens that power the use of its features. It also revealed a sneak peek of the future: **a world running on decentralized applications (Dapps)**. Before we can even fathom what ­Dapps do, we need to be familiar with its underlying technology - the **Blockchain**. A blockchain is a ledger of records organized in ‘blocks’ that are linked together by cryptographic hash validation. Each block typically contains a hash pointer as a link to a previous block, a timestamp and transaction data. It is a digital storage of consensus truth. The key is to understand that this ledger is neither stored in a centralized location nor managed by any single entity, hence its distributed-ness. The block validation system results in new transactions being added irreversibly and old transactions preserved forever for all to see, hence its transparency and resilience.

For an application to be considered a Dapp (pronounced Dee-app) it must meet the following criteria:

1. The application must be completely **open-source,** it must operate autonomously, and with no entity controlling the majority of its tokens. The application may adapt its protocol in response to proposed improvements and market feedback but all changes must be decided by consensus of its users.
2. The application's data and records of operation must be cryptographically **stored in a public, decentralized**[**blockchain**](https://en.bitcoin.it/wiki/Block_chain) in order to avoid any central points of failure.
3. The application must **use a cryptographic token** (ethereum or a token native to its system) which is necessary for access to the application and any contribution of value from (miners / farmers) should be rewarded in the application’s tokens.
4. The application must **generate tokens according to a standard cryptographic algorithm acting as a proof** of the value nodes are contributing to the application (Bitcoin uses the Proof of Work Algorithm).

Ethereum’s white paper, it was stated that the intention of Ethereum is to create an alternative protocol for building decentralized applications. The ownership of the Dapp’s tokens is all that is required for the holder to use the system. It’s that simple. The value of the tokens is determined by how much people value the application. All the incentives, all the monetization, all the ways to raise support are built into this beautifully simple structure.

A typical web application consists of server side code and centralized database that clients can access remotely making it a single point of failure and single authority that controls your data. This is an outdated and has many shortcomings. To overcome this problem we can use Decentralization. **In a decentralized network there is no central server** to which the clients connect, and the data is distributed across nodes in the network. By storing data across its network, the blockchain eliminates the risks that comes with data being held centrally. This is where blockchain has its advantage.

While centralized data is more controllable, information and data manipulation are common. By decentralizing it, **blockchain makes data transparent** to everyone involved. Its network lacks centralized points of vulnerability likewise, it **has no single point of failure**.

Blockchain security methods include the use of **public-key cryptography**. Every node or miner in a decentralized system has a copy of the blockchain. Data quality is maintained by massive database replication and computational trust.

One of the platform that helps us realize Decentralization Applications is **Ethereum**. **Ethereum** is an open-source, public, blockchain-based distributed computing platform and operating system featuring smart contract functionality. **Ethereum** is considered to be one of the pioneer platforms in distributed ledger and blockchain technology. Contracts in Ethereum are written in a high level programming language called – **Solidity**. 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.**

Many uses have been proposed for Ethereum platform, including ones that are impossible or unfeasible. Use case proposals have included finance, the internet-of-things, farm-to-table produce, electricity sourcing and pricing, and sports betting.

Table of Content

[ACKNOWLEDGEMENT 3](#_Toc510007201)

[ABSTRACT 4](#_Toc510007202)

[1. Introdution 8](#_Toc510007203)

[1.1 Objective 8](#_Toc510007204)

[1.2 Scope and Motivation 9](#_Toc510007205)

[2. Decentralized Application 14](#_Toc510007206)

[2.1 History 16](#_Toc510007207)

[2.2 Design and Architecture 20](#_Toc510007208)

[2.3 Applications 22](#_Toc510007209)

[3. System Analysis 25](#_Toc510007210)

[3.1 Existing Systems 25](#_Toc510007211)

[3.1.1 Blockchain 25](#_Toc510007212)

[3.1.2 Ethereum 26](#_Toc510007213)

[3.1.3 Smart Contracts 35](#_Toc510007214)

[3.1.4 Solidity 38](#_Toc510007215)

[3.1.5 JavaScript 40](#_Toc510007216)

[3.1.6 Web3JS 42](#_Toc510007217)

[3.2.7 NPM 43](#_Toc510007218)

[3.1.8 Truffle Framework 44](#_Toc510007219)

[3.1.9 Ganache 45](#_Toc510007220)

[3.1.10 MetaMask 47](#_Toc510007221)

[3.1.11 React 48](#_Toc510007222)

[3.2 Proposed system 50](#_Toc510007223)

[3.2.1 Voting Application 50](#_Toc510007224)

[3.2.2 Real Estate Application 50](#_Toc510007225)

[3.2.3 Blackjack Game 51](#_Toc510007226)

[3.3 System Requirements 52](#_Toc510007227)

[3.3.1 Hardware Requirements 52](#_Toc510007228)

[3.3.2 Software Requirements 52](#_Toc510007229)

[4. System Design 53](#_Toc510007230)

[4.1 Use Case Diagrams 53](#_Toc510007231)

[4.2 Sequence Diagrams 56](#_Toc510007232)

[4.3 Collaboration Diagrams 62](#_Toc510007233)

[5. Implementation 65](#_Toc510007234)

[5.1 Voting Application 67](#_Toc510007235)

[5.2 Real Estate Application 67](#_Toc510007236)

[5.3 Blackjack Game 68](#_Toc510007237)

[6. Output Screenshots 69](#_Toc510007238)

[6.1 Voting Application 69](#_Toc510007239)

[6.2 Real Estate Application 70](#_Toc510007241)

[6.3 Blackjack Game 71](#_Toc510007242)

[7. Conclusion and Future Work 73](#_Toc510007243)

[8. References 74](#_Toc510007244)

# Introdution

Overall Structure of the present environment is centralized just like financial sector business environment that has provided order and standardization to transaction and process which are in place that has worked for decades and may be centuries. Latest blockchain technology is the creation of decentralized app. Unlike today’s common apps, such as Uber or Gmail, dapps don’t require “middlemen” to function. Dapps are revolutionary in that they connect users and providers—customers and businesses—directly, bringing back a true sense of peer-to-peer transactions, trust and transparency. Dapps are also typically open source and don’t have a central point or server that can be vulnerable to system failures or security breaches. Some of the general use cases or fields of D-App are as follows: -

1. A Marketplace That Never Shuts Down (Highly availability of networks).

2. A Secure Marketplace Based on Trust (Transactions are transparent and can be verifies by members of the network)

3. Decentralized to Localized Future

4. Quality assurance

5. Transparency (Technology is almost always open source)

6. Voting System

7. Peer-to-peer global transactions.

8. Smart contracts Transactions

9. Stock exchange

10. Accounting (recording transactions through blockchain)

11. Reduced transaction costs

12. Faster transaction settlements

13. Payment processing: No need to integrate with Stripe or PayPal to accept funds from users. All users can send/receive Ether as a common payment means.

14. User Credentials: Users don't need to sign up; they already have an account, which is a public/private key to bind with their user session and metadata.

15. Database: Storing a lot of data in the blockchain is expensive, so likely the blockchain isn't going to be the *only* database for the application, but mission-critical pieces of data can be stored forever on the blockchain.

16. Logging: Etherum contracts can create their own logs, which a DApp can query to know what's happened in the past, rather than needing to create separate logs.

17. Trust: Your users can trust your code, since not only is the front-end (Javascript) code visible (via browser inspection tools), but the back-end logic (contract code) can be inspected too, so it can be independently verified that your code doesn't have backdoors in it to steal all their funds.

## 1.1 Objective

* To remove the dependency of centralized applications, that includes centralized systems like databases, data centers, server and local internet providers.
* To avoid central point of failures and promote decentralization..
* To bring forth transparency, immutability and security that is provided by decentralized applications.
* To promote open source community by contributing to ethereum.
* To bring forward the idea of decentralization in a world dominated by centralized applcations

Judging by the use cases of a decentralized applications we have decided to make a demonstration of An Online Voting Application, An Online Real Estate and a Blackjack gambling game which is trustworthy, open-source and transparent.

## 1.2 Scope and Motivation

**Voting app**

Voting is the process of choosing someone or something in an election from a group of people which is Auditable transparent secure and accurate Online voting saves our time can increase engagement of large people makes faster result and builds trust as follows:

* **Empowerment.**Voting is the most powerful way for members to have a voice in the leadership and direction of their association. When allowed to vote in fair and open elections, members feel a greater sense of value, ownership, and responsibility. Online elections help empower members of associations, societies, and other democratic organizations by making voting easy and convenient.
* **Accessibility.** Online voting allows association members to access their ballots from anywhere at any time, provided they have an Internet connection. This makes casting a vote convenient and fast. Members can cast their votes from home, from work or “on the go” via their mobile devices.
* **Cost effectiveness.**Online voting reduces election budgets by limiting production costs. Paper, printing, and postage costs are all significantly lower for online elections than for traditional voting methods. In addition, staff will save time because online elections eliminate the need to assemble ballot packages and manually tabulate votes.
* **Security and confidentiality.**A properly designed, secure online voting system has safeguards in place to protect voting information and voter identities. A voting website hosted on a secure server will only be accessible to authorized members through unique voter logins. Online ballots are transmitted from voters’ computers or mobile devices to balloting systems using SSL (Secure Sockets Layer), the same encryption technology used by U.S. financial institutions. These layers of protection form a technology shield that detects unauthorized access, eliminates ballot tampering, and reduces the chance of voting fraud.
* **Eco-friendly.**Web-based balloting conserves resources by reducing the amount of paper associated with an election. Paper ballots, envelopes, flyers, and other paper collateral are either moved to electronic format or eliminated. This is particularly important if your association members are sensitive to the use of natural resources.
* **Membership engagement.**Online voting and other modern participation methods demonstrate an association’s commitment to connecting with its members. It shows the organization is staying relevant and is dedicated to keeping in touch.
* **Appeal to younger members.**According to the Pew Internet and American Life Project, 95 percent of Americans between the ages of 18 and 29 use the Internet. This means that online options such as web voting might be a great way to get younger members involved in the decision-making process.
* **Fast, accurate results.** With online voting there are no rejected, mismarked or invalid ballots. Results are automatically calculated, eliminating the need for manual tabulation and dreaded recounts. Computerized tabulation allows election managers to quickly announce decisions and results.

**Real state app**

Real estate is the property, land, buildings, air rights above the land and underground rights below the land. The term real estate means real, or physical, property. “Real” comes from the Latin root res, or things. Others say it’s from the Latin word rex, meaning “royal,” since kings used to own all land in their kingdoms. The U.S. Constitution initially restricted voting rights to only owners of real estate.

Four Types of Real Estate:

* Residential real estate
* Commercial Real State
* Industrial real state
* Vacant Land real state

Real State Application helps to find out the detail requirement of the house that may includes the desire of the house just like swimming pool attached small playground for children according to the buyer’s requirements he can search the detail of physical property.

**Black jack app**

Blackjack, also known as twenty-one, is a comparing [card game](https://en.wikipedia.org/wiki/Card_game) between usually several players and a dealer, where each player in turn competes against the dealer, but players do not play against each other. It is played with one or more [decks](https://en.wikipedia.org/wiki/Playing_card) of 52 cards, and is the most widely played [casino](https://en.wikipedia.org/wiki/Casino_game) banking game in the world. The objective of the game is to beat the dealer in one of the following ways:

* Get 21 points on the player's first two cards (called a "blackjack" or "[natural](https://en.wikipedia.org/wiki/Natural_(gambling))"), without a dealer blackjack;
* Reach a final score higher than the dealer without exceeding 21; or
* Let the dealer draw additional cards until their hand exceeds 21.

Players are each dealt two cards, face up or down depending on the casino and the table at which you sit. In the U.S., the dealer is also dealt two cards, normally one up (exposed) and one down (hidden). In most other countries, the dealer receives one card face up. The value of cards two through ten is their pip value (2 through 10). Face cards (Jack, Queen, and King) are all worth ten. Aces can be worth one or eleven. A hand's value is the sum of the card values. Players are allowed to draw additional cards to improve their hands. A hand with an ace valued as 11 is called "soft", meaning that the hand will not bust by taking an additional card; the value of the ace will become one to prevent the hand from exceeding 21. Otherwise, the hand is "hard".

Once all the players have completed their hands, it is the dealer’s turn. The dealer hand will not be completed if all players have either busted or received Blackjacks. The dealer then reveals the hidden card and must hit until the cards total 17 or more points. (At most tables the dealer also hits on a "soft" 17, i.e. a hand containing an ace and one or more other cards totaling six.) Players win by not busting and having a total higher than the dealer, or not busting and having the dealer bust, or getting a blackjack without the dealer getting a blackjack. If the player and dealer have the same total (not counting blackjacks), this is called a "push", and the player typically does not win or lose money on that hand. Otherwise, the dealer wins.

Decentralized applications are composed of distributed entities that directly interact with each other and make local autonomous decisions in the absence of a centralized coordinating authority. Such decentralized applications, where entities can join and leave the system at any time, are particularly susceptible to the attacks of malicious entities. Each entity therefore requires protective measures to safeguard itself against these entities. Trust management solutions serve to provide effective protective measures against such malicious attacks. Trust relationships help an entity model and evaluate its confidence in other entities towards securing itself. Trust management is, thus, both an essential and intrinsic ingredient of decentralized applications. However, research in trust management has not focused on how trust models can be composed into a decentralized architecture. The PACE *(*Primary, Alternate, Contingency, and Emergency i.e. technologies or systems of communication*)* architectural style, provides structured and detailed guidance on the assimilation of trust models into a decentralized entity's architecture PACE not only provides an effective and easy way to integrate trust management into decentralized applications, but also facilitates reuse while supporting different types of trust models. Additionally, PACE serves as a suitable platform to aid the evaluation and comparison of trust models in a fixed setting towards providing a way to choose an appropriate model for the setting.

So, which helps us to make our application safe and secure. In Centralized application all data are centrally located so when server is accessed by hacker all the data and information can easily accessed and leaked but in decentralized application data are not stored centrally which makes more secure to data, Payment processing in centralized application need to integrate with Stripe or PayPal to accept funds from users. But in decentralized application all these functionalities are done with the Ether as a common payment means. Users don’t need to sign in / sign up they already have an account which is a public/private key to bind with their user session and metadata. Critical database pieces can be stored with blockchain just like contract codes. Ethereum Contract can create their own logs, in which DApp can query to know what’s happened in the past, rather than needing to create separate logs. Users can trust the code, since not only in the front-end (Java Script) code visible (via browser inspection tools), but the back-end logic (contract code) can be inspected too, so it can be independently verified that code doesn't have backdoors in it to steal all their funds.

In centralized system server machine can face lots of Single Point Failure Limited Scalability as well as Throughput Problems. If the master(server) machine goes down, clients are no longer able to process user requests since the machine that runs the core software to process requests is dead. Now the impact of such a failure will vary from system to system. It will depend on what capabilities are provided by the client component. If clients are not responsible for running any core logic and strongly depend on the server machine, then availability is completely compromised for such a system in case the server machine fails. Because all the core application logic is self-contained in a single server machine, the only way we can imagine to scale the system is **vertical scaling** — by adding more storage, I/O bandwidth, processing power (number of CPUs, number of cores) to the server machine. The more powerful our server machine is, the better the performance of overall system. The problem is that it is difficult to create, maintain and rely on a single magic computer box that solves all of our problems. Eventually the attempts to scale up a single machine by adding more hardware may not turn out to be cost effective throughput as just another side of limited scalability. Imagine there is just a single system, and with a given number of resources and processing power (number of cores etc), there are only so many client requests that can be processed in parallel. During periods of high activity, the single server model will soon turn out to be a bottleneck as it will be difficult to keep up with influx of concurrent user requests, and the number of requests system can really process.

All these problems can be shot out by removing the Centralized system and use of decentralizes System where There is no single server machine that is solely responsible for all the processing. The architecture allows to distribute the workload among multiple compute nodes(aka machines), and each of them is equally capable of servicing requests. There isn’t really any single point of failure because client machines aren’t relying on a single server to fulfill all requests. The system comprises of multiple nodes which might be still available to process user requests. We can scale-out the system by adding more nodes. So Decentralized application helps to solve the critical problems of the centralized system.

# Decentralized Application

Decentralized Applications (dApps) are applications that run on a P2P network of computers rather than a single computer. dApps, have existed since the advent of P2P networks. They are a type of software program designed to exist on the Internet in a way that is not controlled by any single entity.

* Decentralized applications don’t necessarily need to run on top of a blockchain network. Bit Torrent, Popcorn Time, Bit Message, Tor, are all traditional dApps that run on a P2P network, but not on a Blockchain (which is a specific kind of P2P network).
* As opposed to simple smart contracts, in the classic sense of Bitcoin, that sends money from A to B, dApps have an unlimited number of participants on all sides of the market.

DApps are a ‘blockchain enabled’ website, where the Smart Contract is what allows it to connect to the blockchain. The easiest way to understand this is to understand how traditional websites operate.

* 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.

For an application to be considered a dApp in the context of Blockchain, it must meet the following criteria:

* **Application must be completely open-source**It must operate autonomously, and with no entity controlling the majority of its tokens. The application may adapt its protocol in response to proposed improvements and market feedback, but the consensus of its users must decide all changes.
* **Application’s data and records of operation must be cryptographically stored**must be cryptographically stored in a public, decentralized blockchain in order to avoid any central points of failure.
* **Application must use a cryptographic token**(Bitcoin or a token native to its system) which is necessary for access to the application and any contribution of value from (miners/farmers) should be rewarded with the application’s tokens.
* **Application must generate tokens**according to a standard cryptographic algorithm acting as a proof of the value, nodes are contributing to the application (Bitcoin uses the Proof of Work Algorithm).

## 2.1 History

The first work on a cryptographically secured chain of blocks was described in 1991 by Stuart Haber and W. Scott Stornetta. In 1992, Bayer, Haber and Stornetta incorporated Merkle trees to the design, which improved its efficiency by allowing several documents to be collected into one block.

The first blockchain was conceptualized by a person (or group of people) known as Satoshi Nakamoto in 2008. It was implemented the following year by Nakamoto as a core component of the cryptocurrency bitcoin, where it serves as the public ledger for all transactions on the network. Through the use of a blockchain, bitcoin became the first digital currency to solve the double spending problem without requiring a trusted authority and has been the inspiration for many additional applications. The words block and chain were used separately in Satoshi Nakamoto's original paper, but were eventually popularized as a single word, blockchain*,* by 2016.

The computer code behind the decentralized application organization was written by Christoph Jentzsch, and released publicly on GitHub. Simon Jentzsch, Christoph Jentzsch's brother, is also involved in the venture.

The DAO was launched on 30 April 2016 at 01:42:58 AM +UTC on Ethereum Block 1428757, with a website and a 28-day crowdsale to fund the organization. The token sale had raised more than US$34 million by 10 May 2016, and more than US$50 million-worth of Ether (ETH)—the digital value token of the Ethereum network—by 12 May,and over US$100 million by 15 May 2016. On 17 May 2016, the largest investor in the DAO held less than 4% of all DAO tokens and the top 100 holders held just over 46% of all DAO tokens. The fund's Ether value as of 21 May 2016 was more than US$150 million, from more than 11,000 investors.

As of May 2016, The DAO had attracted nearly 14% of all ether tokens issued to date. Since 28 May 2016 the DAO tokens were tradable on various cryptocurrency exchanges.

A paper published in May 2016 noted a number of security vulnerabilities associated with The DAO, and recommended that investors in The DAO hold off from directing The DAO to invest in projects until the problems had been resolved. An Ethereum developer on GitHub pointed out a flaw relating to "recursive calls" in early June that was picked up and blogged by Peter Vessenes, founder of the Blockchain Foundation on June 9, and by June 14, fixes had been proposed and were awaiting approval by members of The DAO.On June 16 further attention was called to recursive call vulnerabilities by bloggers affiliated with the IC3.

On June 17, 2016 The DAO was subjected to an attack that exploited a combination of vulnerabilities, including the one concerning recursive calls, and the user gained control of 3.6 million Ether, around a third of the 11.5 million Ether that had been committed to The DAO; the affected Ether had a value of about $50M at the time of the attack. The funds were put into an account subject to a 28-day holding period under the terms of the Ethereum contract so were not actually gone; members of The DAO and the Ethereum community debated what to do next, with some calling the attack a valid but unethical maneuver, others calling for the Ether to be re-appropriated, and some calling for The DAO to be shut down.

In end of 2016 Poloniex de-listed DAO trading pairs, and in December 2017 Kraken also de-listed the token.

Today blockchain is one of the rising technology. Thousands of companies are under its blessing and many are finding more applications for this.

In its early days, the Web was obviously not as useful as it is today with the array of apps and services that do everything under the sun, but it did have a more DIY distributed feel to it. The Web was pretty decentralized from the outset. The HTTP protocol connected everyone on the planet with a computing device and an Internet connection. In the HTTP protocol guidelines, there are a set of trusted servers that translate the web address you enter into a server address. Furthermore, HTTPS adds another layer of trusted servers and certificate authorities. People would host personal servers for others to connect to, and everyone owned their data. But soon, application servers began taking off and the centralized model of data ownership as we know it today was born. It was the easiest thing to do and it worked. One individual or group pays for maintenance of a server and profits from the users that utilize the software on it. Apps like Myspace and Yahoo! were among the first popular centralized apps.

More recent apps like Uber and Airbnb decentralize the “real-world” parts of a business by providing a central and trusted data store. They are among the first to allow for participation in one moneymaking endeavor from all sides of the economy. Their decentralized business model foreshadows the development of even more decentralized apps.

As the HTTP web grew larger, a new protocol was introduced by a developer named Bram Cohen, called BitTorrent. Bit Torrent was a protocol created as a solution to the lengthy time to download huge media files via HTTP and as an improvement on some of the P2P proposals before it, like Gnutella, Napster, and Grokster. The problem was that downloading huge files took a very long time and as the Web grew, so did the size of files that were available. Meanwhile, hard-drive space was increasing and more people were connected. Bit Torrent solved this by making downloaders into uploaders, as well.

If there was a file you wanted, you would download it from not one, but multiple sources. The more popular the file, the more users who would be downloading it and subsequently uploading it, which meant you would be pulling from multiple sources. The more sources, the faster the download. Seeders were rewarded with faster download speeds, whereas leechers were punished with limited speeds. This tit-for-tat system of transferring data proved to be very useful for large media files like movies and TV shows.

BitTorrent grew and is for many the de facto way to download any sort of large media file like a game or movie. BitTorrent’s speed, resilience, and reward mechanism proved to be better than HTTP for large data sets.

Most likely because of HTTP’s first mover advantage, its infrastructure, and all of the time and money already invested in it. There are currently active projects working on upgrading the HTTP web with BitTorrent-like technology, and they’ll most likely be successful because of BitTorrent’s huge value proposition. As soon as BitTorrent was introduced, developers began to use the technology to create nonprofit decentralized applications.

## 2.2 Design and Architecture

By storing data across its network, the blockchain eliminates the risks that come with data being held centrally. The decentralized blockchain may use ad-hoc message passingand distributed networking.

Its network lacks centralized points of vulnerability that computer crackers can exploit; likewise, it has no central point of failure. Blockchain security methods include the use of public-key cryptography.

A publickey (a long, random-looking string of numbers) is an address on the blockchain. Value tokens sent across the network are recorded as belonging to that address.

A privatekey is like a password that gives its owner access to their digital assets or the means to otherwise interact with the various capabilities that blockchains now support. Data stored on the blockchain is generally considered incorruptible.

This is where blockchain has its advantage. While centralized data is more controllable, information and data manipulation are possible. By decentralizing it, blockchain makes data transparent to everyone involved.

Every node in a decentralized system has a copy of the blockchain. Data quality is maintained by massive database replication and computational trust. No centralized "official" copy exists and no user is "trusted" more than any other. Transactions are broadcast to the network using software. Messages are delivered on a best-effort basis. Mining nodes validate transactions,add them to the block they are building, and then broadcast the completed block to other nodes.

Blockchains use various time-stamping schemes, such as proof-of-work, to serialize changes. Alternate consensus methods include proof-of-stake. Growth of a decentralized blockchain is accompanied by the risk of node centralization because the computer resources required to process larger amounts of data become more expensi,

As depicted in the figure below the front end for a decentralized application is same as a regular web application that depends on a centralized architecture. The main changes are to the backend. Instead of having a server, data warehouse, or a database to store data there is the blockchain itself. This means the data is stored in decentralized network of nodes collectively in blocks and interconnected with each other with a hash.

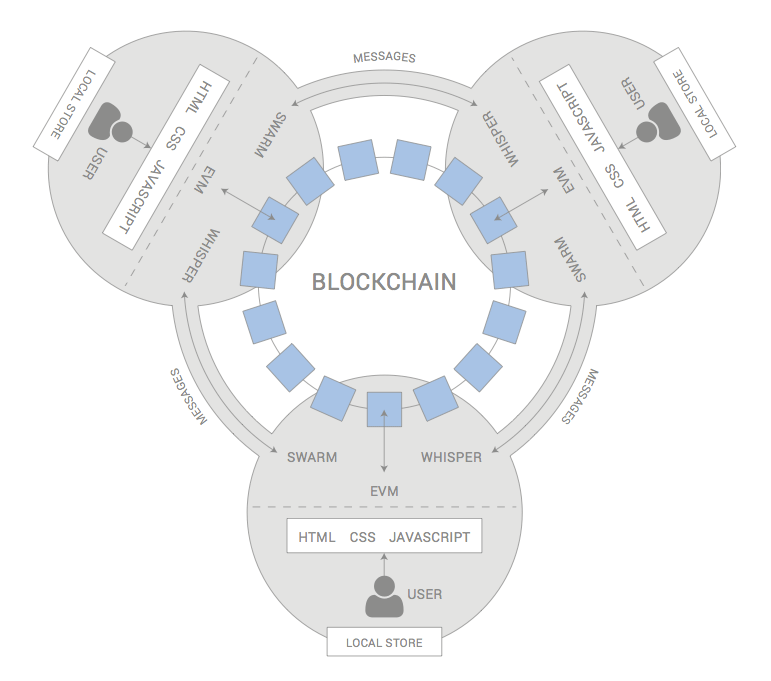


Figure 2.2: Decentralized Applications Architecture

## 2.3 Applications

Ethereum intends to create a protocol for building decentralized applications. Ethereum provides developers with a foundational layer: a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats, and state transition functions. In general, there are three types of applications on top of Ethereum.

* **Financial applications**providing users with more powerful ways of managing and entering into contracts using their money.
* **Semi-financial applications**where money is involved, but there is also a heavy non-monetary side to what is being done
* **Governance Applications**  
  such as online voting & decentralized governance that are not financial at all.

**Examples for such dApps**:

* **Token Systems**  
  On-blockchain token systems have many applications ranging from sub-currencies representing assets such as USD or gold to company stocks, individual tokens representing smart property, secure unforgeable coupons, and even token systems with no ties to conventional value at all, used as point systems for incentivization.
* **Financial derivatives and Stable-Value Currencies**  
  For example, a very desirable application is a smart contract that hedges against the volatility of ether with respect to the US dollar by using the data feed from, e.g., NASDAQ.
* **Identity & Reputation Systems**  
  A contract stating the name of the owner of a land title can be added to the Ethereum network but not modified or removed. Anyone can register a name with some value, and that registration then sticks forever.
* **Decentralized File Storage**  
  A Dropbox-like dApp where a smart contract splits the desired data up into blocks, encrypting each block for privacy, and builds a Merkle tree out of it, then the whole data gets distributed across the network. The most popular implementation to this data is STORJ.
* **Decentralized Autonomous Organizations (DAOs)**  
  A virtual entity that has a certain set of members or shareholders who, perhaps with a 67% majority, have the right to spend the entity’s funds and modify its code. The members would collectively decide on how the organization should allocate its resources.

**Some Example for Decentralization :**-

* Traditional corporations are politically centralized (one CEO), architecturally centralized (one head office) and logically centralized (can’t really split them in half)
* BitTorrent is logically decentralized similarly to how English is. Content delivery networks are similar, but are controlled by one single company.
* Blockchains are politically decentralized (no one controls them) and architecturally decentralized (no infrastructural central point of failure) but they are logically centralized (there is one commonly agreed state and the system behaves like a single computer)

Decentralization , it’s a kind of centralization that is arguably in many cases good (though Juan Benet from IPFS would also push for logical decentralization wherever possible, because logically decentralized systems tend to be good at surviving network partitions, work well in regions of the world that have poor connectivity.

**Three reasons for Decentralization**

The next question is, why is decentralization useful in the first place? There are generally several arguments raised:

* **Fault tolerance** — decentralized systems are less likely to fail accidentally because they rely on many separate components that are not likely relatable to each other. It doesn’t matter if in case some of the nodes in the network fails.
* **Attack resistance** — decentralized systems are more expensive to attack and destroy or manipulate because they lack sensitive central points that can be attacked at much lower cost than the economic size of the surrounding system.
* **Collision resistance**—  it is much harder for participants in decentralized systems to collide to act in ways that benefit them at the expense of other participants,

All three arguments are important and valid, but all three arguments lead to some interesting and different conclusions

# System Analysis

## 3.1 Existing Systems

### 3.1.1 Blockchain

Blockchain is the world's leading software platform for digital assets. Offering the largest production block chain platform in the world, we are using new technology to build a radically better financial system. A **blockchain**,originally **block chain**, is a continuously growing list of records, called *blocks*, which are linked and secured using cryptography.

 Each block typically contains a cryptographic hash of the previous block, a timestamp and transaction data. By design, a blockchain is inherently resistant to modification of the data. It is "**an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way**".

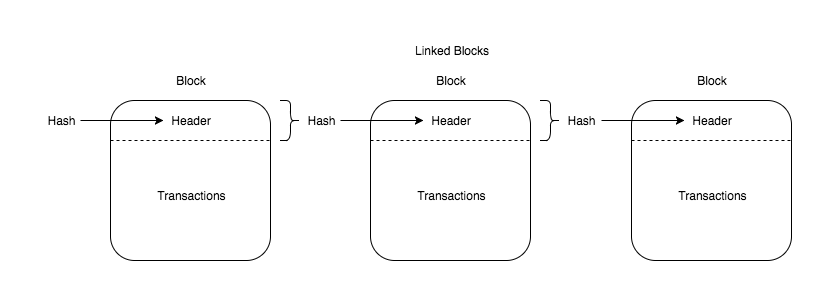


Figure 3.1.1: Blocks in a Blockchain.

For use as a distributed ledger, a blockchain is typically managed by a peer-to-peer network collectively adhering to a protocol for validating new blocks. Once recorded, the data in any given block cannot be altered retroactively without the alteration of all subsequent blocks, which requires collusion of the network majority. Blockchains are secure by design and exemplify a distributed computing system with high Byzantine fault tolerance.

Decentralized consensus has therefore been achieved with a blockchain. This makes blockchains potentially suitable for the recording of events, medical records, and other records management activities, such as identity management, transaction processing, documenting provenance, food traceability or voting.

Blockchain was invented by Satoshi Nakamoto in 2008 for use in the cryptocurrency bitcoin, as its public transaction ledger.[1] The invention of the blockchain for bitcoin made it the first digital currency to solve the double spending problem without the need of a trusted authority or central server.

Blockchain has moved into the mainstream, with a wide variety of organizations that are keen to explore how it can be implemented for their businesses.

In August 2014, the bitcoin blockchain file size, containing records of all transactions that have occurred on the network, reached 20 GB (gigabytes). In January 2015, the size had grown to almost 30 GB, and from January 2016 to January 2017, the bitcoin blockchain grew from 50 GB to 100 GB in size. The words *block* and *chain* were used separately in Satoshi Nakamoto's original paper, but were eventually popularized as a single word, *blockchain,* by 2016.

Whether Blockchain technology is still in its early stages or if it is ready to be implemented, one thing for sure is that opinions on it are still being formed by the community. As we see large implementations of Blockchain in various forms either in government or industry.

### 3.1.2 Ethereum

The intent of Ethereum is to merge together and improve upon the concepts of scripting, altcoins and on-chain meta-protocols, and allow developers to create arbitrary consensus-based applications that have the scalability, standardization, feature-completeness, ease of development and interoperability offered by these different paradigms all at the same time. Ethereum does this by building what is essentially the ultimate abstract foundational layer: a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions.

A bare-bones version of Namecoin can be written in two lines of code, and other protocols like currencies and reputation systems can be built in under twenty. Smart contracts, cryptographic "boxes" that contain value and only unlock it if certain conditions are met, can also be built on top of our platform, with vastly more power than that offered by Bitcoin scripting because of the added powers of Turing-completeness, value-awareness, blockchain-awareness and state.

**Ethereum Accounts**

In Ethereum, there are objects called "accounts", with each account having a 20-byte address and state transitions being direct transfers of value and information between accounts. An Ethereum account contains four fields:

* The nonce, a counter used to make sure each transaction can only be processed once
* The account's current ether balance
* The account's contract code, if present
* The account's storage (empty by default)

"Ether" is the main internal crypto-fuel of Ethereum, and is used to pay transaction fees. In general, there are two types of accounts: externally owned accounts, controlled by private keys, and contract accounts, controlled by their contract code. An externally owned account has no code, and one can send messages from an externally owned account by creating and signing a transaction; in a contract account, every time the contract account receives a message its code activates, allowing it to read and write to internal storage and send other messages or create contracts in turn.

**Messages and Transactions**

"Messages" in Ethereum are somewhat similar to “transactions” in Bitcoin, but with three important differences. First, an Ethereum message can be created either by an external entity or a contract, whereas a Bitcoin transaction can only be created externally. Second, there is an explicit option for Ethereum messages to contain data. Finally, the recipient of an Ethereum message, if it is a contract account, has the option to return a response; this means that Ethereum messages also encompass the concept of functions.

The term "transaction" is used in Ethereum to refer to the signed data package that stores a message to be sent from an externally owned account. Transactions contain the recipient of the message, a signature identifying the sender, the amount of ether and the data to send, as well as two values called STARTGAS and GASPRICE.

In order to prevent exponential blowup and infinite loops in code, each transaction is required to set a limit to how many computational steps of code execution it can spawn, including both the initial message and any additional messages that get spawned during execution. STARTGAS is this limit, and GASPRICE is the fee to pay to the miner per computational step. If transaction execution "runs out of gas", all state changes revert - except for the payment of the fees, and if transaction execution halts with some gas remaining then the remaining portion of the fees is refunded to the sender. There is also a separate transaction type, and corresponding message type, for creating a contract; the address of a contract is calculated based on the hash of the account nonce and transaction data. An important consequence of the message mechanism is the "first class citizen" property of Ethereum - the idea that contracts have equivalent powers to external accounts, including the ability to send message and create other contracts. This allows contracts to simultaneously serve many different roles

For example,

One might have a member of a decentralized organization (a contract) be an escrow account (another contract) between an paranoid individual employing custom quantum-proof Lamport signatures (a third contract) and a co-signing entity which itself uses an account with five keys for security (a fourth contract).

**Ethereum State Transition Function**

The Ethereum state transition function, APPLY(S,TX) -> S' can be defined as follows:

1. Check if the transaction is well-formed (ie. has the right number of values), the signature is valid, and the nonce matches the nonce in the sender's account. If not, return an error.

2. Calculate the transaction fee as STARTGAS \* GASPRICE, and determine the sending address from the signature. Subtract the fee from the sender's account balance and increment the sender's nonce. If there is not enough balance to spend, return an error.

3. Initialize GAS = STARTGAS, and take off a certain quantity of gas per byte to pay for the bytes in the transaction.

4. Transfer the transaction value from the sender's account to the receiving account. If the receiving account does not yet exist, create it. If the receiving account is a contract, run the contract's code either to completion or until the execution runs out of gas.

5. If the value transfer failed because the sender did not have enough money, or the code execution ran out of gas, revert all state changes except the payment of the fees, and add the fees to the miner's account.

6. Otherwise, refund the fees for all remaining gas to the sender, and send the fees paid for gas consumed to the miner.

For example,

Suppose that the contract's code is: msg.sender

if (! contract.storage[msg.data[0]])

contract.storage [msg.data [0]] = msg.data [1]

Note that in reality the contract code is written in the low-level EVM code; this example is written in Solidity, one of the high-level language, for clarity, and can be compiled down to EVM code. We use this programming language throughout the project. Suppose that the contract's storage starts off empty, and a transaction is sent with 10 ether value, 2000 gas, 0.001 ether gas price, and two data fields: [ 2, 'CHARLIE' ] [3] . The process for the state transition function in this case is as follows:

1. Check that the transaction is valid and well formed.

2. Check that the transaction sender has at least 2000 \* 0.001 = 2 ether. If it is, then subtract 2 ether from the sender's account.

3. Initialize gas = 2000; assuming the transaction is 170 bytes long and the byte-fee is 5, subtract 850 so that there is 1150 gas left.

4. Subtract 10 more ether from the sender's account, and add it to the contract's account.

5. Run the code. In this case, this is simple: it checks if the contract's storage at index 2 is used, notices that it is not, and so it sets the storage at index 2 to the value CHARLIE. Suppose this takes 187 gas, so the remaining amount of gas is 1150 - 187 = 963

6. Add 963 \* 0.001 = 0.963 ether back to the sender's account and return the resulting state.

If there was no contract at the receiving end of the transaction, then the total transaction fee would simply be equal to the provided GASPRICE multiplied by the length of the transaction in bytes, and the data sent alongside the transaction would be irrelevant.

Additionally, note that contract-initiated messages can assign a gas limit to the computation that they spawn, and if the sub-computation runs out of gas it gets reverted only to the point of the message call.

Hence, just like transactions, contracts can secure their limited computational resources by setting strict limits on the sub-computations that they spawn.

**Code Execution**

The code in Ethereum contracts is written in a low-level, stack-based bytecode language, referred to as "Ethereum virtual machine code" or "EVM code". The code consists of a series of bytes, where each byte represents an operation. In general, code execution is an infinite loop that consists of repeatedly carrying out the operation at the current program counter (which begins at zero) and then incrementing the program counter by one, until the end of the code is reached or an error or STOP or RETURN instruction is detected. The operations have access to three types of space in which to store data:

* The stack, a last-in-first-out container to which 32-byte values can be pushed and popped
* Memory, an infinitely expandable byte array
* The contract's long-term storage, a key/value store where keys and values are both 32 bytes. Unlike stack and memory, which reset after computation ends, storage persists for the long term.

The code can also access the value, sender and data of the incoming message, as well as block header data, and the code can also return a byte array of data as an output.

The formal execution model of EVM code is surprisingly simple. While the Ethereum virtual machine is running, its full computational state can be defined by the tuple (block\_state, transaction, message, code, memory, stack, pc, gas), where block\_state is the global state containing all accounts and includes balances and storage. Every round of execution, the current instruction is found by taking the pc-th byte of code, and each instruction has its own definition in terms of how it affects the tuple.

**Applications**

In general, there are three types of applications on top of Ethereum. The first category is financial applications, providing users with more powerful ways of managing and entering into contracts using their money. This includes sub-currencies, financial derivatives, hedging contracts, savings wallets, wills, and ultimately even some classes of full-scale employment contracts. The second category is semi-financial applications, where money is involved but there is also a heavy non-monetary side to what is being done; a perfect example is self-enforcing bounties for solutions to computational problems. Finally, there are applications such as online voting and decentralized governance that are not financial at all.

**Token Systems** On-blockchain token systems have many applications ranging from sub-currencies representing assets such as USD or gold to company stocks, individual tokens representing smart property, secure unforgeable coupons, and even token systems with no ties to conventional value at all, used as point systems for incentivization.

Token systems are surprisingly easy to implement in Ethereum. The key point to understand is that all a currency, or token system, fundamentally is a database with one operation: subtract X units from A and give X units to B, with the proviso that (i) X had at least X units before the transaction and (2) the transaction is approved by A. All that it takes to implement a token system is to implement this logic into a contract.

**Further Applications**

1. **Savings wallets**.

Suppose that Alice wants to keep her funds safe, but is worried that she will lose or someone will hack her private key. She puts ether into a contract with Bob, a bank, as follows:

* Alice alone can withdraw a maximum of 1% of the funds per day.
* Bob alone can withdraw a maximum of 1% of the funds per day, but Alice has the ability to make a transaction with her key shutting off this ability.
* Alice and Bob together can withdraw anything.

Normally, 1% per day is enough for Alice, and if Alice wants to withdraw more she can contact Bob for help. If Alice's key gets hacked, she runs to Bob to move the funds to a new contract. If she loses her key, Bob will get the funds out eventually. If Bob turns out to be malicious, then she can turn off his ability to withdraw

2. **Crop insurance**.

One can easily make a financial derivatives contract but using a data feed of the weather instead of any price index. If a farmer in Iowa purchases a derivative that pays out inversely based on the precipitation in Iowa, then if there is a drought, the farmer will automatically receive money and if there is enough rain the farmer will be happy because their crops would do well.

3. **A decentralized data feed**.

For financial contracts for difference, it may actually be possible to decentralize the data feed via a protocol called "SchellingCoin". SchellingCoin basically works as follows: N parties all put into the system the value of a given datum (eg. the ETH/USD price), the values are sorted, and everyone between the 25th and 75th percentile gets one token as a reward. Everyone has the incentive to provide the answer that everyone else will provide, and the only value that a large number of players can realistically agree on is the obvious default: the truth. This creates a decentralized protocol that can theoretically provide any number of values, including the ETH/USD price, the temperature in Berlin or even the result of a particular hard computation.

4. **Smart multi-signature escrow**.

Bitcoin allows multisignature transaction contracts where, for example, three out of a given five keys can spend the funds. Ethereum allows for more granularity; for example, four out of five can spend everything, three out of five can spend up to 10% per day, and two out of five can spend up to 0.5% per day. Additionally, Ethereum multisig is asynchronous - two parties can register their signatures on the blockchain at different times and the last signature will automatically send the transaction.

5. **Cloud computing**.

The EVM technology can also be used to create a verifiable computing environment, allowing users to ask others to carry out computations and then optionally ask for proofs that computations at certain randomly selected checkpoints were done correctly. This allows for the creation of a cloud computing market where any user can participate with their desktop, laptop or specialized server, and spot-checking together with security deposits can be used to ensure that the system is trustworthy (ie. nodes cannot profitably cheat). Although such a system may not be suitable for all tasks; tasks that require a high level of inter-process communication, for example, cannot easily be done on a large cloud of nodes. Other tasks, however, are much easier to parallelize; projects like SETI@home, folding@home and genetic algorithms can easily be implemented on top of such a platform.

6. **Peer-to-peer gambling**.

Any number of peer-to-peer gambling protocols, such as Frank Stajano and Richard Clayton's Cyberdice, can be implemented on the Ethereum blockchain. The simplest gambling protocol is actually simply a contract for difference on the next block hash, and more advanced protocols can be built up from there, creating gambling services with near-zero fees that have no ability to cheat.

7. **Prediction markets**.

Provided an oracle or SchellingCoin, prediction markets are also easy to implement, and prediction markets together with SchellingCoin may prove to be the first mainstream application of futarchy as a governance protocol for decentralized organizations. 8. On-chain decentralized marketplaces, using the identity and reputation system as a base.

The Ethereum protocol was originally conceived as an upgraded version of a cryptocurrency, providing advanced features such as on-blockchain escrow, withdrawal limits and financial contracts, gambling markets and the like via a highly generalized programming language.

The Ethereum protocol would not "support" any of the applications directly, but the existence of a Turing-complete programming language means that arbitrary contracts can theoretically be created for any transaction type or application. What is more interesting about Ethereum, however, is that the Ethereum protocol moves far beyond just currency.

Protocols and decentralized applications around decentralized file storage, decentralized computation and decentralized prediction markets, among dozens of other such concepts, have the potential to substantially increase the efficiency of the computational industry, and provide a massive boost to other peer-to-peer protocols by adding for the first time an economic layer.

Finally, there is also a substantial array of applications that have nothing to do with money at all. The concept of an arbitrary state transition function as implemented by the Ethereum protocol provides for a platform with unique potential; rather than being a closed-ended, single-purpose protocol intended for a specific array of applications in data storage, gambling or finance, Ethereum is open-ended by design, and we believe that it is extremely well-suited to serving as a foundational layer for a very large number of both financial and non-financial protocols in the years to come.

### 3.1.3 Smart Contracts

A Smart Contracts is a computer code running on top of a blockchain containing a set of rules under which the parties to that smart contract agree to interact with each other. If and when the pre-defined rules are met, the agreement is automatically enforced. The smart contract code facilitates, verifies, and enforces the negotiation or performance of an agreement or transaction. It is the simplest form of decentralized automation.

It is a mechanism involving digital assets and two or more parties, where some or all of the parties deposit assets into the smart contract and the assets automatically get redistributed among those parties according to a formula based on certain data, which is not known at the time of contract initiation.

A smart contract can only be as smart as the people coding taking into account all available information at the time of coding.

While smart contracts have the potential to become legal contracts if certain conditions are met, they should not be confused with legal contracts accepted by courts and or law enforcement. However, we will probably see a fusion of legal contracts and smart contracts emerge over the next few years as the technology becomes more mature and widespread and legal standards are adopted.

**Characteristics of a Smart Contract**

Smart contracts are capable of tracking performance in real time and can bring tremendous cost savings. Compliance and controlling happen on the fly. In order to get external information, a smart contract needs information oracles, which feed the smart contract with external information.

**Smart Contracts**are

* Self-verifying
* Self-executing
* Tamper resistant

**Smart Contracts** can

* Turn legal obligations into automated processes.
* Guarantee a greater degree of security.
* Reduce reliance on trusted intermediaries.
* Lower transaction costs.
* Act like intelligence in blockchain

**How smart contracts work**

It's worth noting that bitcoin was the first to support basic smart contracts in the sense that the network can transfer value from one person to another. The network of nodes will only validate transactions if certain conditions are met.

But, bitcoin is limited to the currency use case.

By contrast, ethereum replaces bitcoin's more restrictive language (a scripting language of a hundred or so scripts) and replaces it with a language that allows developers to write their own programs.

Ethereum allows developers to program their own smart contracts, or 'autonomous agents', as the ethereum white paper calls them. The language is 'Turing-complete', meaning it supports a broader set of computational instructions.

Smart contracts can:

* Function as 'multi-signature' accounts, so that funds are spent only when a required percentage of people agree
* Manage agreements between users, say, if one buys insurance from the other
* Provide utility to other contracts (similar to how a software library works)
* Store information about an application, such as domain registration information or membership records.The main advantages of smart contracts
* agent neutrality in signing deals;
* automation in signing deals, time saving: excludes human participation in transactions, everything is done by the prescribed program code;
* safety: data in the decentralized registry cannot be lost and cyber attacked;
* precision: no mistakes can be made due to the absence of hand-filled forms;

Smart contracts are in demand in various fields.

*In gambling businesses*, it is possible to provide information about the total entry and supposed score through the smart contract, and, which is important, the algorithm of this contract will be available for anyone to check. If the player wins, the money is automatically transferred to his/her account, if the player loses, the money is sent back to the bookmaker’s office account.

*In logistics*, blockchain-based smart contracts can shorten the chain of third party agents, speed up delivery, reduce the price for the consumer and excludes the possibility of theft due to the fact that all parties have access to the electronic system that controls all the processes. Applying smart contracts *in elections*excludes any manipulation of the third party because it is not possible to divert the source code.

Like any other technology, smart contracts have their own disadvantages:

* The consumers are quite suspicious because it is a new technology and they do not understand it yet.
* Making changes. For example, you may change your mind about renting an apartment, but the data is already registered and it is technically difficult to make corrections. This may bring mistakes into the system and make it less safe.
* One can keep and save data in smart contracts safely and it is void of any distortions, only if the code is written perfectly and precisely. Humans can be tired or make clerical errors and thus the whole system is endangered.
* The third party agents do not disappear but start playing a different role. The need for lawyers experienced in IT increases in the future because the programmers of smart contracts will need consultations for making new kinds of contracts.

**Smart Contract Coding: -**

Solidity is a smart contract programming language. The syntax is similar to that of JavaScript, and it is designed to compile to code for the Ethereum Virtual Machine, to create contracts for voting, crowdfunding, blind auctions, multi-signature wallets and more.

### 3.1.4 Solidity

Solidity is a contract-oriented, high-level language for implementing smart contracts. This platform has similar syntax to the scripting language of JavaScript. Solidity as a programming language is made to enhance the Ethereum Virtual Machine. Solidity is statically typed scripting language which does the process of verifying and enforcing the constraints at compile-time as opposed to run-time. It was influenced by C++, Python and JavaScript and is designed to target the Ethereum Virtual Machine (EVM). Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features. It is possible to create contracts for voting, crowdfunding, blind auctions, multi-signature wallets and more. Solidity is designed to develop Smart Contracts in Ethereum Blockchain platform.

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. In each Contract, we can define State Variables, Methods, and Events etc. This contract can manage transactions between blocks in Blockchain network. Each block has a particular address in the form of a cryptographic key that generated by the result of some functions including hashing of adjacent blocks. This creates a strong relationship between adjacent blocks. So that manipulation or any other form of hacking in nodes or blocks are not easy or not even possible. Solidity is one of the many languages that can be used to develop EVM (Ethereum Virtual Machine) understandable bytecode. There are many built-in classes and Libraries in Solidity which support hassle free smart contract development.

Following are some of the features of solidity which are very similar to common high-level languages like Java and C++.

* **Statically typed Language**

Though it is having a structure of JavaScript, unlike JavaScript it is a Statically Typed language. For example, you must declare the type of a variable like in C++ and Java before it is used. Otherwise, a compile-time error will be generated.

* **Contract and Interfaces**

‘Contract’ is a unique data structure of Solidity language, it helps to create and manage contracts easily. Contracts can be inherited by child Contracts and can create complex contract structures.

* **Function Modifier**

It is similar to function override in other OOP languages. Suppose you want to execute a function in a different way when a condition is met. In this case, you can use function modifier feature to change the behavior of the function. Generally useful in inherited Contracts to override the parent function behavior.

* **Events**

Events are used to write information from contract to Blockchain log. The event is similar to a function which takes the data as the argument and writes to Blockchain client’s log.

* **Access Specifies**

This is similar to the access specifies in other OOP languages like private and public. In Solidity the name and access rights have some change.

* **Explicit Type conversion**

You can perform explicit conversion of different data types. Such conversions are generally checked at compile time, but there are exceptions also.

* **Memory Arrays**

Dynamic arrays can be directly allocated to memory.

* **Import**

With ‘import’ keyword you can import other source files to your contract.

These are some very basic features you can find in Solidity, refer more and understand the simplicity of Solidity.

### 3.1.5 JavaScript

JavaScript is a cross-platform, object-oriented scripting language used to make webpages interactive (e.x. having complex animations, clickable buttons, popup menus, etc.).  There are also more advanced server side versions of javascript such as Node.Js which allow us to add more functionality to a website than simply downloading files (such as realtime collaboration between multiple computers). Inside a host environment (for example, a web browser), JavaScript can be connected to the objects of its environment to provide programmatic control over them.

We can say that JavaScript is the third layer of the layer cake of standard web technologies, two of which is HTML & CSS.

* HTML is the markup language that we use to structure and give meaning to our web content, for example defining paragraphs, headings, and data tables, or embedding images and videos in the page.
* CSS is a language of style rules that we use to apply styling to our HTML content, for example setting background colors and fonts, and laying out our content in multiple columns.
* JavaScript is a scripting language that enables us to create dynamically updating content, control multimedia, animate images, and pretty much everything else. (Okay, not everything, but it is amazing what we can achieve with a few lines of JavaScript code.)

JavaScript contains a standard library of objects, such as Array, Date, and Math, and a core set of language elements such as operators, control structures, and statements. Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects; for example:

* *Client-side JavaScript* extends the core language by supplying objects to control a browser and its Document Object Model (DOM). For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.
* *Server-side JavaScript* extends the core language by supplying objects relevant to running JavaScript on a server. For example, server-side extensions allow an application to communicate with a database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.
* This means that in the browser, JavaScript can change the way the webpage (DOM) looks. And, likewise, Node.js JavaScript on the server can respond to custom requests from code written in the browser.

**Advantages of JavaScript**

* **Less server interaction** − We can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
* **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
* **Increased interactivity** − We can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
* **Richer interfaces** − We can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

**Limitations of JavaScript**

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features −

* Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
* JavaScript cannot be used for networking applications because there is no such support available.
* JavaScript doesn't have any multithreading or multiprocessor capabilities.

### 3.1.6 Web3JS

**It’s easier to identify the major differences between Web 1.0** (users passively consult web pages and for the most part don’t participate in generating content) **and Web 2.0** (users create content and interact with sites and with each other through social media, forums, etc.). **Instead, with Web 3.0, the differences are not as clearly defined.** The **term, coined by the reporter John Markoff of The New York Times in 2006, refers to a new evolution of the web,** its third generation, and includes specific innovations and practices.

Below are 5 main features that can help us define Web 3.0:

**1) Semantic Web**  
The next evolution of the Web involves the Semantic Web. The semantic web improves web technologies in order to **generate, share and connect content through search and analysis based on the ability to understand the meaning of words**, rather than on keywords or numbers.

**2) Artificial Intelligence**  
Combining this capability with natural language processing, in Web 3.0, **computers can understand information like humans in order to provide faster and more relevant results**. They become more intelligent to satisfy the needs of users.

**3) Connectivity**  
With Web 3.0, **information is more connected thanks to semantic metadata**. As a result, the user experience evolves to another level of connectivity that leverages all the available information.

**4) Ubiquity**  
**Content is accessible by multiple applications**, every device is connected to the web, the services can be used everywhere.

***How Web 3.0 can change our lives***

**This scenario of Web 3.0 is not a dream… For the most part, it’s already a reality today (for example the semantic web and artificial intelligence) thanks to cognitive technology.** Using semantic analysis and natural language processing, the Cogito cognitive technology helps you understand meaning and extract insight from web content and unstructured data.

### 3.2.7 NPM

**npm** is a package manager for the java script programming language. It is the default package manager for the JavaScript runtime environment Node.js. It consists of a command line client, also called npm, and an online database of public and paid-for private packages, called the npm registry. The registry is accessed via the client, and the available packages can be browsed and searched via the npm website. The package manager and the registry are managed by npm, Inc.

**History NPM :-**

npm is written entirely in JavaScript and was developed by Isaac Z. Schlueter as a result of having "seen module packaging done terribly" and with inspiration from the shortcomings of other similar projects such as PEAR (PHP) and CPAN (Perl)

**Description :-**

npm is included as a recommended feature in [Node.js](https://en.wikipedia.org/wiki/Node.js) installer. npm consists of a [command line](https://en.wikipedia.org/wiki/Command-line_interface) client that interacts with a remote registry. It allows users to consume and distribute JavaScript modules that are available on the registry. Packages on the registry are in CommonJS format and include a metadata file in JSON format. Over 477,000 packages are available on the main npm registry. The registry has no vetting process for submission, which means that packages found there can be low quality, insecure, or malicious. Instead, npm relies on user reports to take down packages if they violate policies by being low quality, insecure or malicious. npm exposes statistics including number of downloads and number of depending packages to assist developers in judging the quality of packages.

npm can manage packages that are local dependencies of a particular project, as well as globally-installed JavaScript tools. When used as a dependency manager for a local project, npm can install, in one command, all the dependencies of a project through the package.json file. In the package.json file, each dependency can specify a range of valid versions using the semantic versioning scheme, allowing developers to auto-update their packages while at the same time avoiding unwanted breaking changes. npm also provides version-bumping tools for developers to tag their packages with a particular version. npm also provides package-lock.json file which has the entry of the exact version used by the project after evaluating semantic versioning in package.json.

### 3.1.8 Truffle Framework

Truffle is a development environment, testing framework and asset pipeline for Ethereum, aiming to make life as an Ethereum developer easier. It is one of the most widely used IDEs in the Ethereum community. Developers can use it to build and deploy DApps for testing purposes with many features that make it more attractive to users with a Web 3.0 dev background. With Truffle, we get:

* 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

Anyone who wants to dive into Ethereum development and needs a framework so they can better organize their DApp development assets and not have to worry about manually setting up a test environment.

**Installing Truffle Dependencies**

You’re going to need to have installed node.js. From there, we need to install truffle

***(npm install -g truffle)*.**

Also, you’re going to need to run a local blockchain RPC server to test and develop against. I recommend using testrpc, which we install by running

***npm install -g ethereumjs-testrpc.***

Next let’s make sure we have our **testrpc** running in the background. Open our terminal and run the command testrpc. That's all! It runs on port 8545by default, just like most Ethereum RPCs, and so does Truffle.

### 3.1.9 Ganache

**Ganache** is a personal blockchain for **Ethereum** development you can use to deploy contracts, develop your applications, and run tests. It is available as both a desktop application as well as a command-line tool (formerly known as the TestRPC). **Ganache** is available for Windows, Mac, and Linux

**Installation**

* **Windows:**  Ganache-\*.appx
* **Mac:** Ganache-\*.dmg
* **Linux:** Ganche-\*.AppImage

**Ganache** CLI :-

We can use Command line Version (formerly known as TestRPC), we can get it through npm i.e

npm install -g ganache-cli

**Main Interface :-**

When we launch Ganache, the screen will shows some details about the server, and also list out a number of accounts. Each accounts is given 100 ether Having ether automatically an all accounts allows us to focus on developing our application

**There we can see four pages: -**

1. The accounts page shows the accounts generated and their balances. This is the default view.
2. The Blocks page shows each block as mined on the blockchain, along with gas used and transactions.
3. The Transaction page lists all transaction run against the blockchain.
4. The Logs shows for the server, Which is useful for debugging

**Ganache Settings : -**

1. The Server page page shows detail about the network connection, including hostname, port, network ID, and whether to automatically mine each transaction into a block.
2. The Accounts & Keys page sets details about the number of accounts created, and whether o use a specific mnemonic or let ganache its own.
3. The Chain page sets details about the actual working of the generated blockchain, including gas limit and gas price.
4. The Advanced page toggles Google Analytics, which is useful for the Ganche team to track uses of the application.

### 3.1.10 MetaMask

MetaMask is a bridge that allows us to visit the distributed web of tomorrow in our browser today. It allows us to run Ethereum dApps right in our browser without running a full Ethereum node. In simple word it injects the web3 into our app environment.

The regular web puts Amazon’s servers at the focus: Our browser makes HTTP requests to it for the webpage and sends our payment details to it. However, Ethereum is decentralized. To pay you would have to write our transaction on the Ethereum blockchain through any Ethereum node (could be on our computer or elsewhere) and broadcast it across the network. Amazon would then look at the blockchain to check that Ether has been transferred from our account into its own.

At the moment, regular web browsers don’t know how to connect to an Ethereum node and read or write to the Ethereum blockchain so you would have to start up our Mist browser and fiddle with little details like eth addresses to make the payment. This is where MetaMask comes in.

Metamask includes a secure identity vault, providing a user interface to manage our identities on different sites and sign blockchain transactions.

MetaMask injects a JavaScript library called web3.js into the namespace of each page our browser loads. web3.js is written by the Ethereum core team and has functions that regular webpages can use to make read and write requests on the blockchain that are consistent with the existing protocol.

Furthermore, MetaMask allows users to specify which Ethereum node to send these requests to. The ability to send requests to nodes outside of the user’s computers is important because it means that people can use Ethereum without having to download a node consisting the 10+GB blockchain on to their computers.

### 3.1.11 React

React is a JavaScript tool that makes it easy to reason about, construct, and maintain stateless and stateful user interfaces. It provides the means to declaratively define and divide a UI into UI component (i.e. react components) using HTML-like nodes called React nodes. React nodes eventually get transformed into a format for UI rendering (e.g., HTML/DOM, canvas, svg, etc.).

React is a component-based library which is used to develop interactive UI’s (*User Interfaces*). It is currently one of the most popular JavaScript front end libraries which has a strong foundation and a large community supporting it.

In ReactJS, everything is a component. Consider one lego house as an entire application. Then compare each of the lego block to a component which acts as a building block. These blocks components are integrated together to build one bigger and dynamic application.

The biggest advantage of using components is that, you can change any component at any point of time without affecting the rest of the applications. This feature is most effective when implemented with larger and real time applications where data changes frequently. Each time any data is added or updated, ReactJS automatically updates the specific component whose state has *actually* changed. This saves the browser from the task of reloading the whole application to reflect the changes.

ReactJS was developed by *Jordan Walke*, a software engineer working at Facebook. Facebook implemented ReactJS in 2011 in its *newsfeed* section, but it was released to the public in May, 2013. After the implementation of ReactJS, Facebook’s UI underwent drastic improvement. This resulted in satisfied users and a sudden boost in its popularity.

**Features of React :-**

**JSX:**JSX stands for JavaScript XML. It’s an XML/ HTML like syntax used by React. It extends the ECMAScript so that XML/ HTML like text can co-exist along with JavaScript react code. This syntax is used by the pre-processors like *Babel* to transform HTML like text found in JavaScript files into standard JavaScript objects. With JSX, we can go a step further by again embedding the HTML code inside the JavaScript. This makes HTML codes easy to understand and boosts JavaScript’s performance while making our application robust.

**Virtual DOM**: Like an actual DOM, *virtual DOM* is also a node tree that lists the elements and their attributes and content as Objects and their properties. Reacts render function creates a node tree out of the react components. Then, it updates this tree in response to the mutations in data model caused by various actions done either by the user or by the system.

**Testability**: React views can be used as functions of the state (state is an object which determines how component will render and behave). Thus, we can easily manipulate with state of the components which we pass to the ReactJS view and take a look at the output and triggered actions, events, functions, etc. This makes React applications quite easy to test and debug.

**Server-Side Rendering(SSR):**Server-Side rendering allows you to pre-render the initial state of our react components at the server side only. With SSR, the server’s response to the browser becomes only the HTML of the page which is now ready to be rendered. Thus, the browser can now start rendering without having to wait for all the JavaScript to be loaded and executed. As a result, the webpage loads faster. Here the user will be able to see the web page inspite of React still downloading the JavaScript, creating the virtual DOM, linking events, etc. at the back end.

**One-Way Data Binding:**Unlike other frameworks, ReactJS follows unidirectional data flow or one-way data binding. Major advantage of One-Way-Data binding is that, throughout the application the data flows in a single direction which gives you better control over it. Because of this, application’s state is contained in specific stores and as a result, rest of the components remains loosely coupled. This makes our application more flexible leading to increased efficiency.

**Simplicity:**Use of JSX files makes the application really simple and easy to code as well as understand. Even though we can use plain JavaScript here, using JSX is easier. React’s component-based approach along with distinct life cycle methods also make it simple to learn.

**Learning Curve:**Learning curve of React is quite low as compared to other frameworks. Anyone having even basic knowledge in programming can easily learn React. So, if you have previous knowledge in HTML and JavaScript you can get on our hands on it very quickly.

## 3.2 Proposed system

### 3.2.1 Voting Application

The traditional voting system that is still in use is unreliable, inconsistent, slow and non-transparent. This is a fairly old way of doing things. The online counterpart is not very secure as there is a presence of central server and the database to store the data is also centralized. This mean there is a single point of failure. In order to solve these problems we propose our decentralized voting application where data and the code is store in the blockchain itself making it decentralized. The candidates and their respective votes are permanently stored in the blockchain and is immutable. This gives us more secure scenario than using a centralized server. Our smart contracts verifies user’s wallet address and only allows one wallet address to vote once. The ethereum blockchain also maintains a transaction logs that one can use to trace back to a vote given by a voter through his/her wallet address. After the election date expires the application automatically generates results and winner’s name.

### 3.2.2 Real Estate Application

Use of Decentralized Real Estate on Blockchain helps us to remove fraud, forgery, middleman and brokers. Using Blockchain we can replace paper proof with digital proof such as proof of existence, proof of ownership, proof of transaction, proof of exchange, proof of value. These all can represent that a valid data is present in the blockchain. It eliminates the costs associated with title insurance and fraud. The use of blockchain also sacks the middleman which only exist because he/she holds information that is not publicly accessible easily. With a public blockchain the information is divided to everyone and is free and publicly available. Including all of these we can create a safe environment for buyers and sellers to transfer funds to each other using our decentralized application in exchange for the legal documents for the digital proof.

### 3.2.3 Blackjack Game

Most gambling that are organized in casinos comes with several trade-offs like extra charges for exchanges, game cheats or even matching up against professional dealers. Our application has open source algorithm which can be verified by anyone before they start playing. There is un-biased gameplay for all the members. This genuinely depends on one’s luck and is very accessible from anyplace.

The cash exchange can be done right after winning without any extra charges for exchanges. The transferred transaction can be traced back through a log and if there are any failures, there is a fallback function implemented that will automatically handle most of the error cases. This will bring real money valued gambling to online world where people win or lose real assets just like in a real gambling game.

The smart contract prohibits anyone to accidently transfer funds to the dealer without losing and the dealer will also not be able to transfer anything without the client’s win. Also, the contract verifies that there is no cheating in the client’s end.

## 3.3 System Requirements

### 3.3.1 Hardware Requirements

Like every web application the hardware requirements for decentralized applications are no different. The base hardware specification for the project are:

* Minimum Pentium-IV with Dual Core Processor
* RAM 1 GB and Above
* HDD 80 GB Hard Disk Space and Above with SCSI
* Keyboard, Mouse, Monitor.

### 3.3.2 Software Requirements

The list of applications for achieving this project are mentioned below:

* Solidity (contract-oriented programming language)
* JavaScript – Web3JS.
* Ganache or TESTRPC (Local Blockchain Environment)
* MetaMask (Web3 Injection Chrome/ FireFox Extension)
* Truffle Framework
* Node-Package-Manager (NPM)
* ReactJS Library (FrontEnd)
* Browser (Chrome, Firefox , Safari, IE, Opera etc. - Anyone)

# System Design

## 4.1 Use Case Diagrams

**4.1.1 Voting Application**

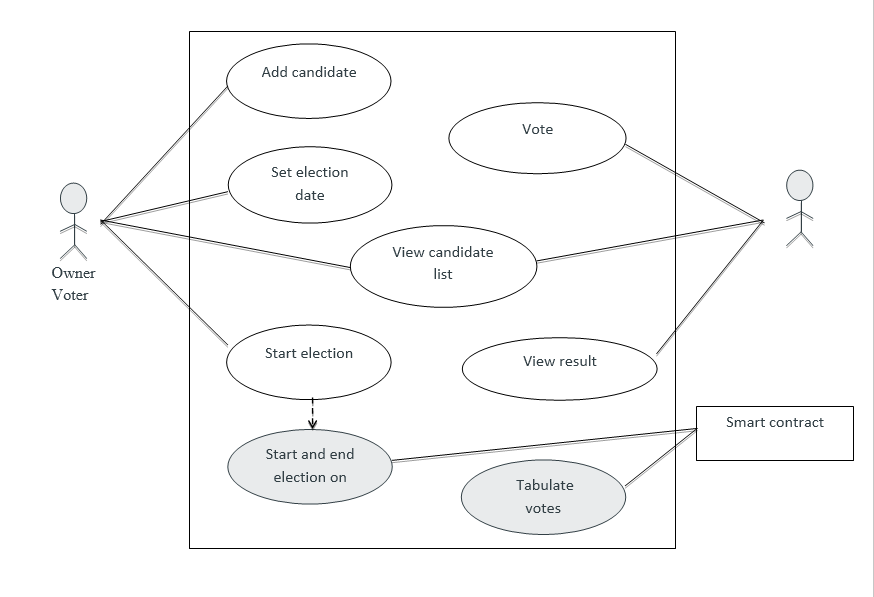


Figure 4.1.1: Use Case Diagram for Voting Application

**4.1.2 Real Estate Application**

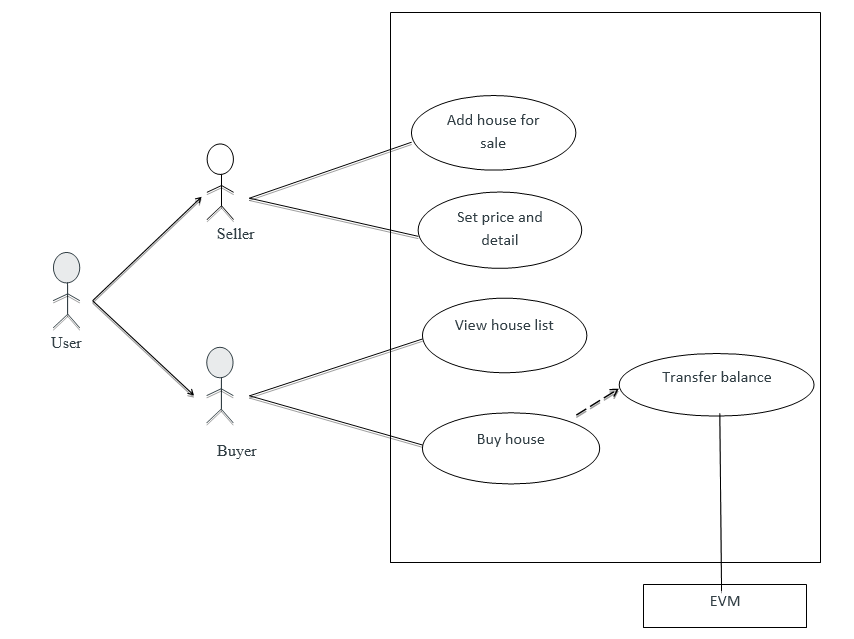


Figure 4.1.2: Use Case Diagram for Real Estate Application

**4.1.3 Blackjack Game**

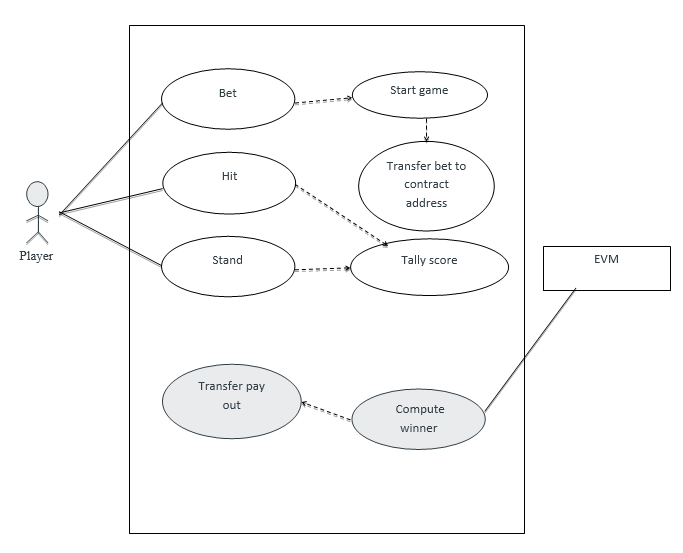


Figure 4.1.3 : Use Case Diagram for Blackjack Game

## 4.2 Sequence Diagrams

**4.2.1 Voting Application**

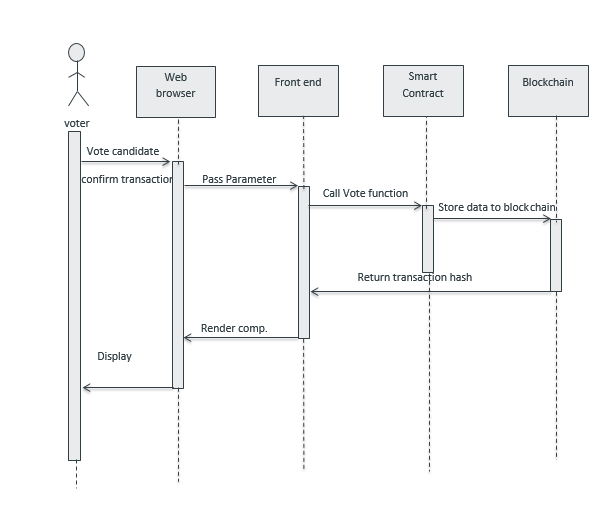
****

Figure 4.2.1: Sequence Diagram for Voter in Voting Application

**4.2.2 Real Estate Application**

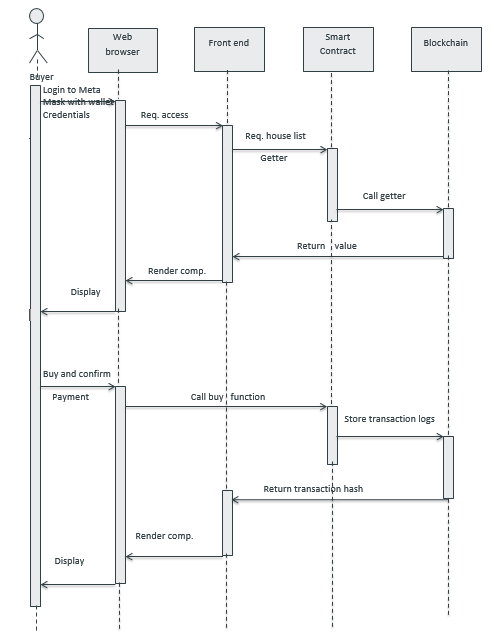


Figure 4.2.2 (a): Sequence Diagram for Buyer in Real Estate Application

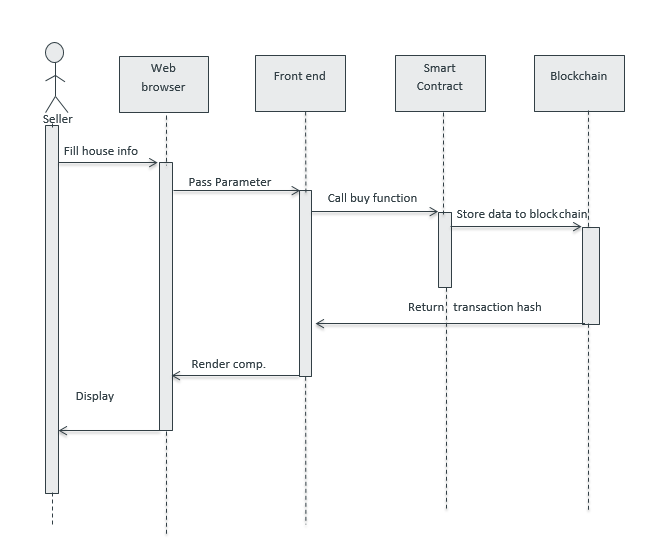


Figure 4.2.2 (b): Sequence Diagram for Seller in Real Estate Application

**4.2.3 Blackjack Game**

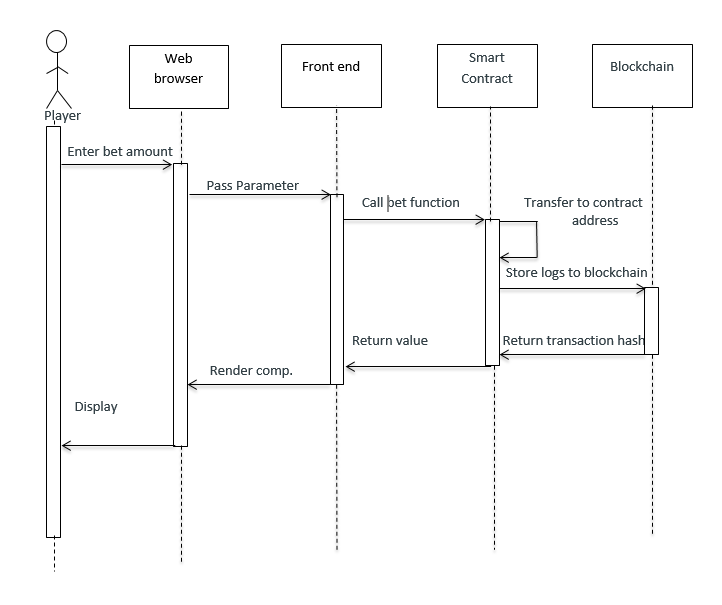


Figure 4.2.3 (a): Sequence Diagram for Player in Blackjack Game

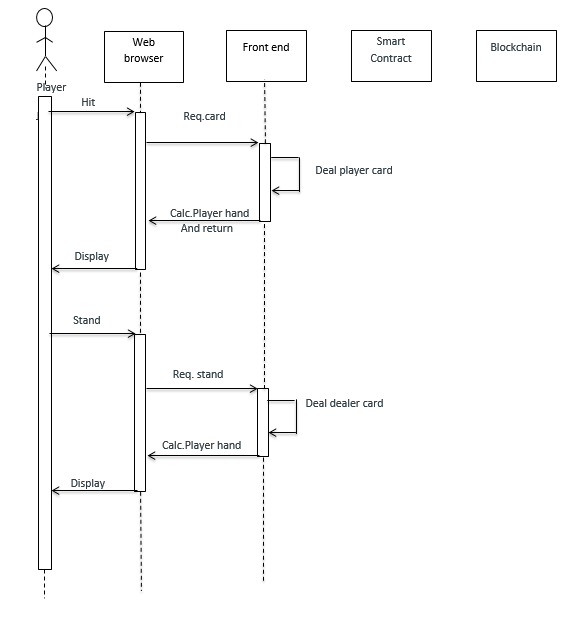


Figure 4.2.3 (b) : Sequence Diagram for Player Decision in Blackjack Game

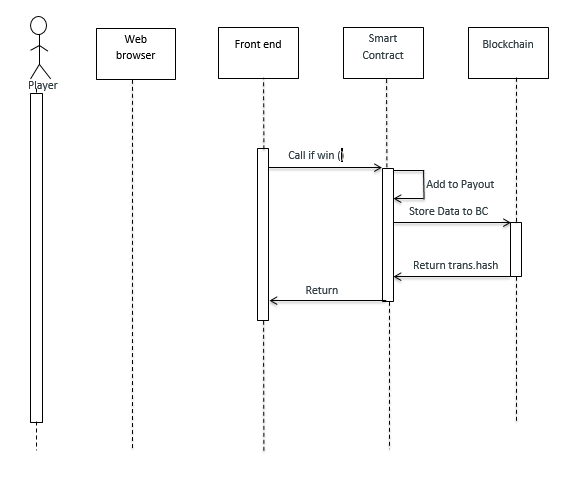


Figure 4.2.3 (c): Sequence Diagram for Win in Blackjack Game

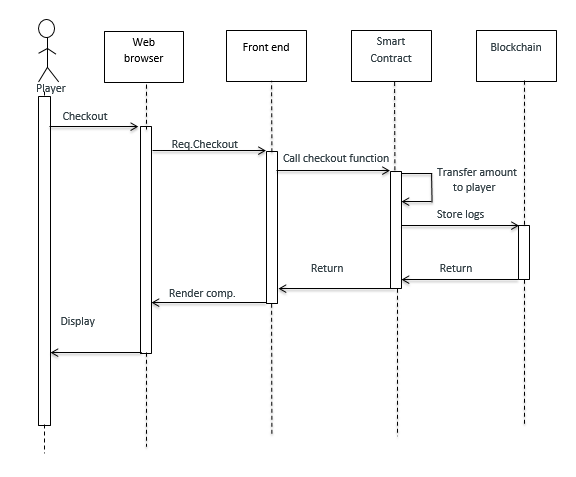


Figure 4.2.3 (d): Sequence Diagram for Checkout in Blackjack Game

## 4.3 Collaboration Diagrams

**4.3.1 Voting Application**

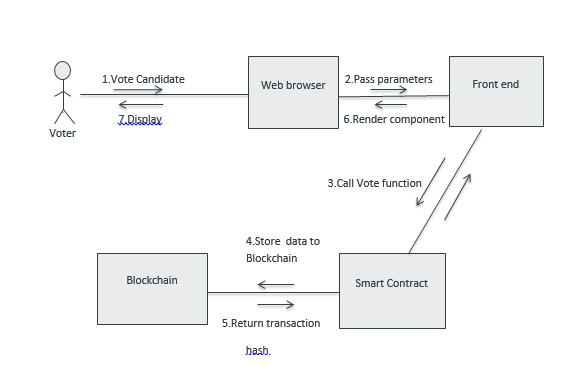


Figure 4.3.1: Collaboration Diagram for Voter in Voting Application

**4.3.2 Real Estate Application**

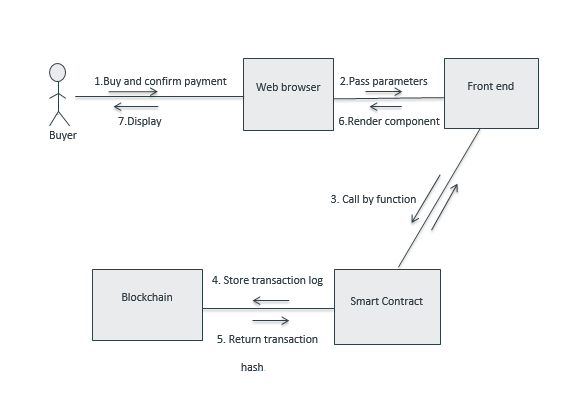


Figure 4.3.2 (a): Collaboration Diagram for Buyer in Real Estate Application

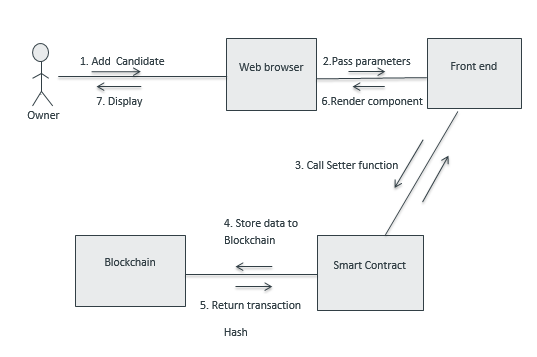


Figure 4.3.2 (b): Collaboration Diagram for Owner in Real Estate Application

**4.3.3 Blackjack Game**

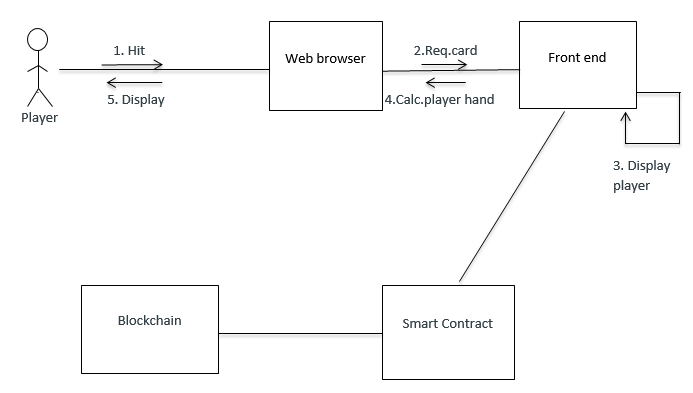


Figure 4.3.3 (a): Collaboration Diagram for Game Actions in Blackjack Game

Figure 4.3.3 (b): Collaboration Diagram for Start a Blackjack Game

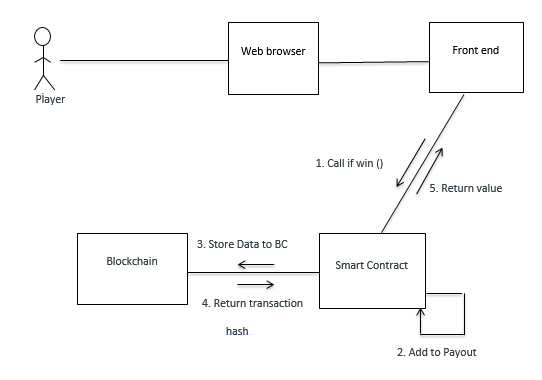
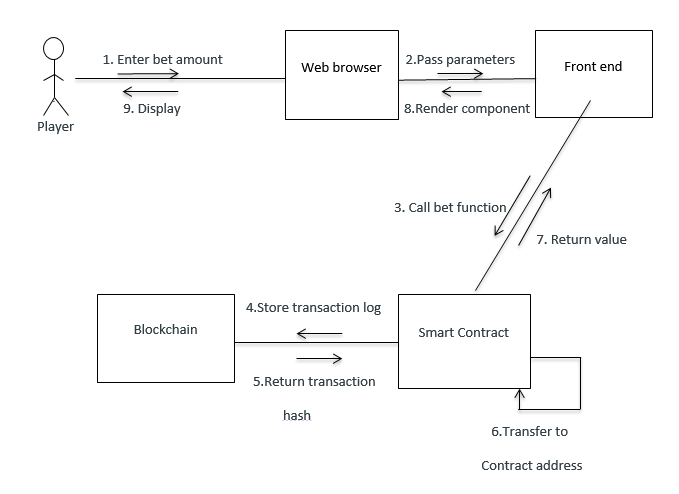


Figure 4.3.3 (c) : Collaboration Diagram for Win in Blackjack Game

# Implementation

Blockchain has the potential to enhance almost every area of business. While the technology is still in its infancy, experts emphasize some of blockchain’s low-hanging fruit as finance, supply chain management, accounting, auditing, manufacturing, and healthcare.

The movement of goods, financing, and documentation across international borders can be complex and place burdens on centralized parties. Blockchain, when integrated with systems, IoT devices and customer-facing interfaces, can help facilitate global transactions with limited intervention and secure record keeping. Implementing blockchain in supply chain practices will also help improve accountability of suppliers and vendors and product safety for consumers.

Blockchain can ensure security in any instance where sensitive information needs to be transferred, stored, or verified. EY emphasizes that blockchain peer-to-peer networks could potentially replace clearinghouses. Additional benefits include standardizing record-keeping, streamlining auditing and compliance, all of which will ultimately lead to organizational savings.

Blockchain data storage will become a massive disruptor shortly. (3-5 years) Current cloud storage services are centralized — thus you the users must place trust in a single storage provider. “They” control all of your online assets. On the other hand, with the Blockchain, this can become decentralized. For instance, Storj is beta-testing cloud storage using a Blockchain-powered network to improve security and decrease dependency. Additionally, users (you) can rent out their excess storage capacity, Airbnb-style, creating new marketplaces. Anyone on the internet can store your data at a pre-agreed price. Hashing and having the data in multiple locations are the keys to securing it. Storj.io and factom are two start-ups exploring this idea. After encrypting your data, it is sent out to a network with easy to track basic metadata.

Smart contracts are legally binding programmable digitized contracts entered on the blockchain. What developers do is to implement legal contracts as variables and statements that can release of funds using the bitcoin network as a ‘3rd party executor’, rather than trusting a single central authority.

For example, if two people want to exchange $100 at a specific time in future when a set of preconditions are met, the conditions, payout, and parties’ details would be programmed into a smart contract. Once the defined conditions are met, funds would be released and sent to the appropriate party as per terms.

By giving computers control over contracts, we can make business more efficient and make the legal system more equitable.

Smart contracts solve the problem of intermediary trust between parties to an agreement, whether that is between people transferring assets like gold, or executing decisions between two parties in a betting contract. Smart contracts are digital which are embedded with an if-this-then-that (IFTTT) code, which gives them self-execution. In real life, an intermediary ensures that all parties follow through on terms. The blockchain not only waives the need for third parties, but also ensures that all ledger participants know the contract details and that contractual terms implement automatically once conditions are met. We can use smart contracts for all sort of situations, such as financial derivatives, insurance premiums, property law, and crowd funding agreements, among others. Some of the other implementation are as follows:

## 5.1 Voting Application

One of the field of blockchain field is Real estate, UBITQUITY offers a simple user experience for securely recording, tracking, and transferring deeds with its SaaS blockchain platform. The company is helping real estate, title, and mortgage companies benefit from a clean record of ownership, reducing future title search time, increased confidence, and transparency.

UBITQUITY’s platform is meant to be a parallel recording and tracking system to the current legacy paper one. When transactions occur all relevant information about the property are put on the platform. The company also launched its private alpha platform for secure document storage/integration within and outside the United States.

Silvertown helps housing associations and large property managers monitor the vital signs of their assets using smart home technologies.

## 5.2 Real Estate Application

The greatest barrier to getting electoral processes online, according to its detractors, is security. Using the blockchain, a voter could check that her or his vote was successfully transmitted while remaining anonymous to the rest of the world. In 2014, Liberal Alliance, a political party in Denmark, became the first organization to use blockchain to vote. With American voter turnout still shockingly low, distributed digital voting may represent a way to enfranchise non-participants. Blockchain technology can be used for voter registration and identity verification, and electronic vote counting to ensure that only legitimate votes are counted, and no votes are changed or removed. Creating an immutable, publicly-viewable ledger of recorded votes would be a massive step toward making elections more fair and democratic.

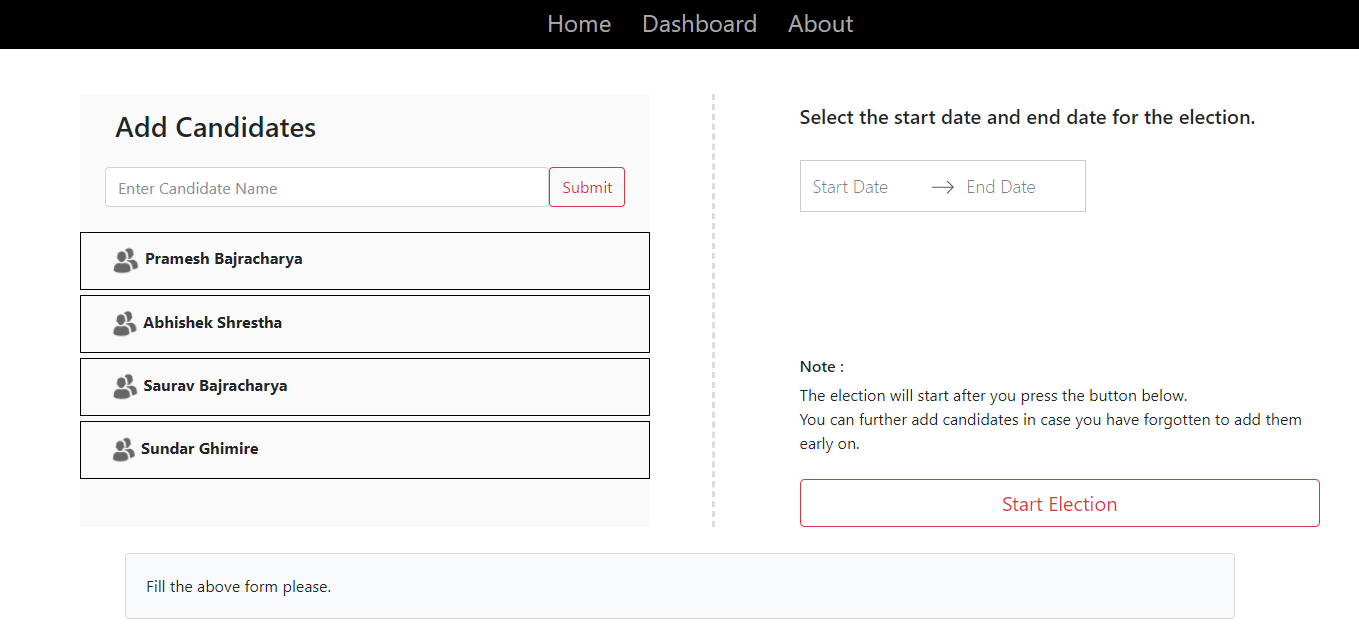
Examples: - Democracy Earth and Follow My Vote are two startups aiming to disrupt democracy itself through creating blockchain-based online voting systems for governments.

## 5.3 Blackjack Game

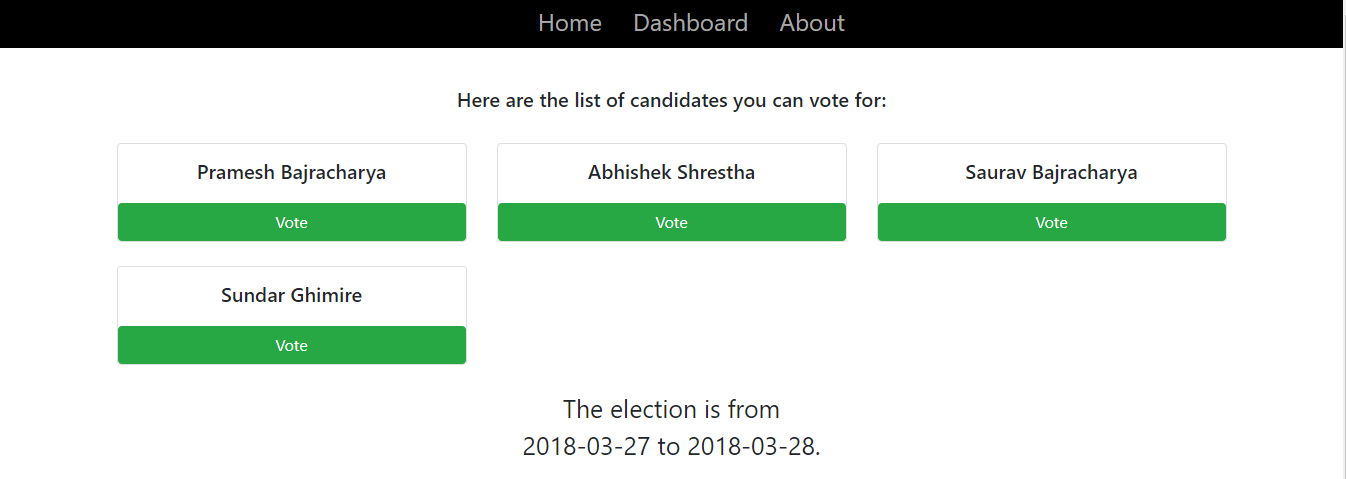
Blockchain even found its fans in gaming and gambling industries with some colorful examples, proving that there are no limits to the fantasy of imaginative entrepreneurs. Etheria is a virtual world in which players can own tiles, farm them for blocks and build things. The entire state of the world is held in and all player actions are made through the decentralized, trustless Ethereum blockchain. Made possible by smart contracts and oracles in the blockchain, the First Blood is a platform that lets eSports players challenge each other on the field and win rewards. The company has built a decentralized eSports reward platform that every gamer loves. Etheramid claims to be the most honest social invitation game we will ever see because no one can change the logic. Even the owner or developer. It’s an Ethereum algorithm based on contract verified by Ether. Camp. It’s an invitation-only game that rewards gamers with Ether from each invited participant up to 7 levels. The idea is to invite and build our own Etheramid network. The FreeMyVunk Movement (the ReVUNKolution) is dedicated to empowering people to fight for ownership rights over their virtual property, virtual junk or VUNK. As for gambling, some examples: include CoinPalace, Etheroll, Rollin, Ethereum Jackpot and more.

# Output Screenshots

## 6.1 Voting Application

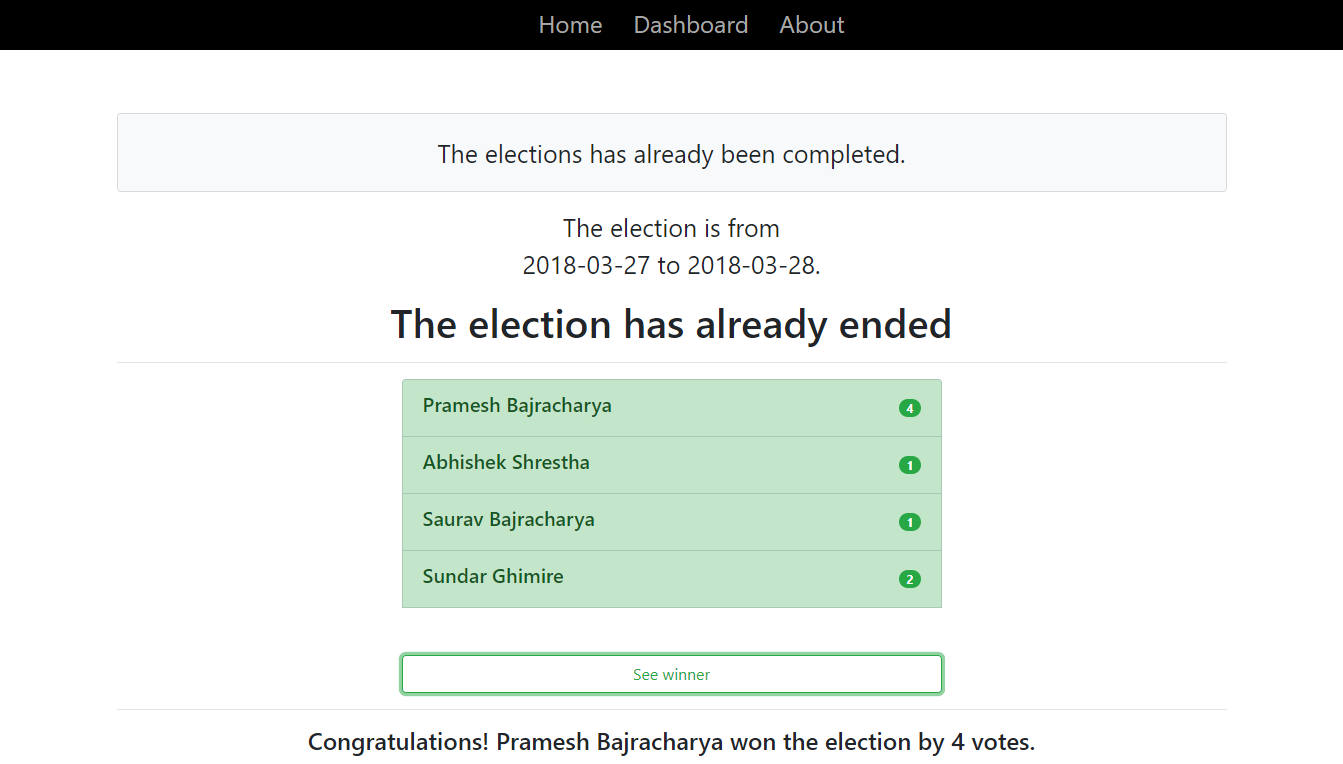


Screenshot 6.1 (a): Adding candidates and starting voting phase.



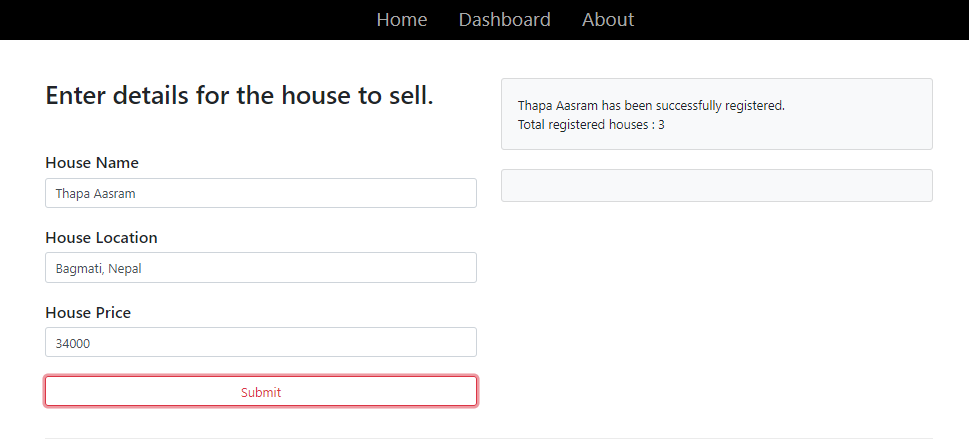
Screenshot 6.1 (b) : Voters selecting candidates to vote

## 

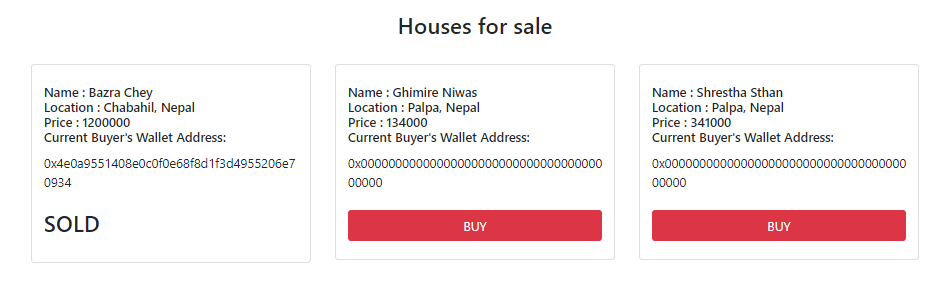


Screenshot 6.1 (c) : Results and Winner Declaration for the Voting app

## 6.2 Real Estate Application

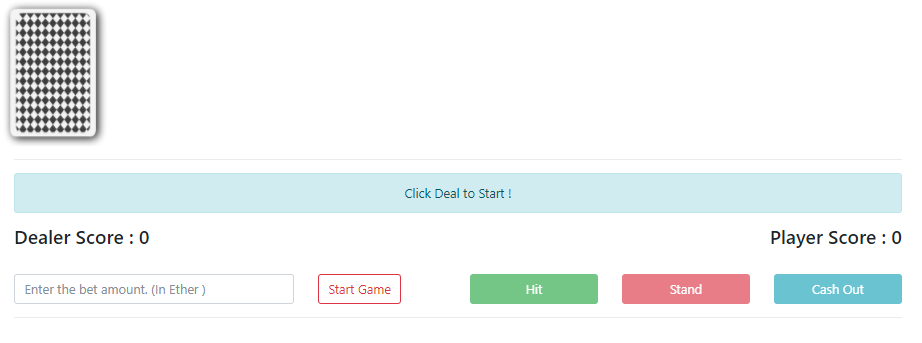


Screenshot 6.2 (a): Adding House for sale.



Screenshot 6.2 (b): List of house to buy.

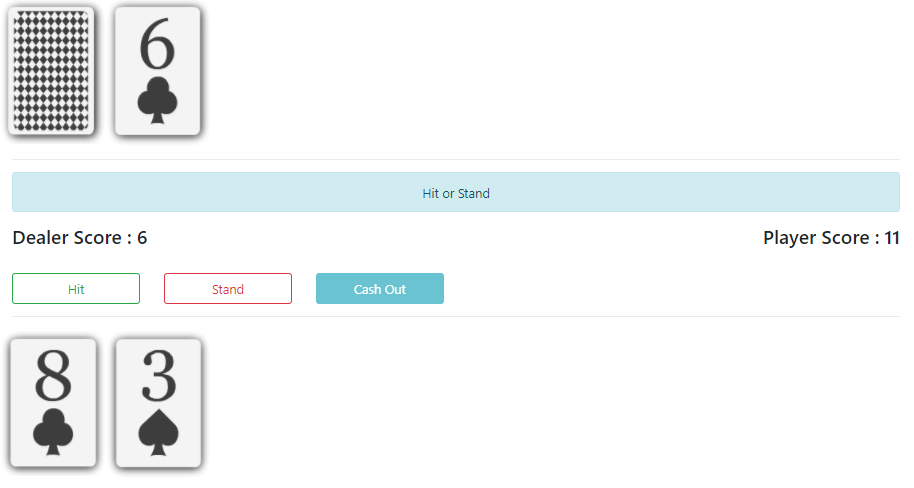
## 6.3 Blackjack Game



Screenshot 6.3 (a): Starting Blackjack.



Screenshot 6.3 (a): Blackjack Losing State.



Screenshot 6.3 (a): Blackjack Winning State.

# Conclusion and Future Work

Most of the decentralized applications that are made on top of blockchain are still underdeveloped. They are still in their infancy. The future potential of the blockchain applications is still unravelling. The next couples of years will be all about experimenting and applying to all aspects of society and improving this technology to greater extent.

Regardless of which application comes first on a global scale. It’s important to note that for the blockchain to work, the node-to-node network must be motivated and agree to work under ethical standards. Once, and only if, these standards are adhered to, the blockchain could become a powerful tool for improving business, conducting fair trade, democratizing the global economy, and helping support more open and fair societies.

The Ethereum ecosystem will continue to expand as Ethereum gets under the mainstream radar. The recent rise in the price of Ether have brought along a new wave of interest in decentralized application.

Our plan includes further extension on this project that will include very strict authentication and user verification on the larger scale. The biometrics mapping with address stored in the blockchain will open up huge list of possibilities. This can be beneficial in each and every sector. Taking advantage of this immutability governmental fields and medical sectors can have a much needed change. Blockchain will help to address several modern-day security concerns, including issues with contracts, identity, and fraud management. Blockchain-based lists will allow online retailers and financial organizations to conveniently vet their customers and fight against fraudulent activities.

We plan on making this set of apps available on the live ethereum network and testnet as well. The code is already open-sourced in GitHub and can be viewed by anyone at the very moment.

# References

* http://ieeexplore.ieee.org/document/1310705/
* https://www.quora.com/How-does-centralized-and-decentralized-computing-differ
* https://www.fool.com/investing/2017/12/11/5-big-advantages-of-blockchain-and-1-reason-to-be.aspx
* https://www.entrepreneur.com/article/306420
* https://www.huffingtonpost.com/entry/3-ways-decentralized-apps-are-revolutionizing-online\_us\_59dc7e60e4b060f005fbd69e
* http://www.socialfish.org/2012/08/online-voting-advantages-and-challenges-for-associations
* https://docs.npmjs.com/files/package.json
* https://www.edureka.co/blog/what-is-react/
* https://blockchainhub.net/smart-contracts/
* https://en.wikipedia.org/wiki/Smart\_contract
* https://www.coindesk.com/information/ethereum-smart-contracts-work/
* http://www.expertsystem.com/web-3-0/
* https://1stwebdesigner.com/what-is-web-3-0/
* https://www.lifewire.com/what-is-web-3-0-3486623
* https://www.prokarma.com/blog/2014/10/16/what-exactly-web-30
* https://www.huffingtonpost.com/ameer-rosic-/5-blockchain-applications\_b\_13279010.html
* https://blockgeeks.com/guides/blockchain-applications/
* <https://futurethinkers.org/industries-blockchain-disrupt/>
* Book: Frontier Ethereum Guide – By ETH Dev and Ethereum Community, Decentralized Applications – Siraj Raval