**DETAILED REPORT ON ETHEREUM BLOCKCHAIN USE CASES**

* Ethereum is a decentralized blockchain platform that establishes a peer-to-peer network that securely executes and verifies application code, called smart contracts. Smart contracts allow participants to transact with each other without a trusted central authority
* Ethereum (ETH) is the second leading form of cryptocurrency behind Bitcoin (BTC). Unlike Bitcoin, it does not have a maximum supply.
* Ever since Ethereum was introduced in the blockchain space with its presence in 2015, Vitalik Buterin, a Canadian-Russian programmer, has brought forth new decentralized applications (dApps). However, Ethereum’s success is significantly attributed to the implementation of smart contracts.
* The Ethereum Blockchain is a potential distributed infrastructure that facilitates you to complete projects using smart contracts.Ethereum possesses the following functionalities:

1. **Create your cryptocurrencies**Ethereum, you can create a tradable token that you can use as a new currency. Tokens created with the Ethereum platform use a standard coin API to be compatible with any Ethereum blockchain wallet.
2. **Develop virtual organizations**You can write a smart contract to build a blockchain-based organization. You can then add more people to your organization and enable voting rules. Members of your organization can vote and if it reaches the required number of votes, the smart contract executes automatically.
3. **Build dApps**Ethereum allows developers to develop secure and fault-tolerant decentralized apps that eliminate intermediaries and offer transparency.
4. **Raise funds**You can also use Ethereum smart contracts for fundraising. With Ethereum, you can write a smart contract and a deadline. In case you fail to accomplish the goal, all donations will automatically be reimbursed to donors without disputes or commissions

**TOOLS AND TECHNOLOGIES REQUIRED FOR IMPLEMENTING ETHEREUM SMART CONTRACTS**

**Truffle**

Truffle is an Ethereum development framework that allows developers to write and test smart contracts. Written in JavaScript, Truffle contains a compiler for the Solidity programming language. Truffle Contract is a JavaScript library that allows importing of compiled smart contracts.

**Web3.js**

It is an Ethereum JavaScript API that interacts with the Ethereum network via RPC calls.

Visual Studio Code

A functional code editor.

**Ganache CLI**

It is an Ethereum remote procedure call client within the Truffle framework that is also known as TestRPC.

**Parity**

It is a secure and fast Ethereum client for handling Ethereum accounts and tokens.

**Node.js**

It is a javascript runtime environment used for server-side programming. Node.js is required to test the Ethereum smart contract’s functionality while ensuring its secure and proper operation. You need to install a package manager, for example, Yarn along with Node.js.

# **STEPS TO CREATE, TEST AND DEPLOY ETHEREUM SMART CONTRACTS**

1. **Create a Wallet at CryptoShare.App**

* Visit <https://www.cryptoshareapp.com> in your Chrome browser and enable it. Once it is enabled, click on its icon on the top right of the browser page. Clicking on it will open it in a new tab of the browser.
* Click on Create Wallet and agree to the terms and conditions by clicking I agree to proceed further. It will ask you to create a password.
* After you create a password, it will send you a secret backup phrase that can be used for backing up and restoring the account. Do not disclose it or share it with someone, as this phrase can take away your Ethers.

**2.Select a Test Network**

You might also find the following test networks in your wallet:

Robsten Test Network

Kovan Test Network

Rinkeby Test Network

Goerli Test Network

* The above networks are for testing purposes only; note that the Ethers of these networks have no real value.

**Step 3: Add Some Dummy Ethers in Your Wallet**

* In case you want to test the smart contract, you must have some dummy Ethers in your cryptoshare.app wallet.
* For example, if you want to test a contract using the Robsten test network, select it, and you will find 0 ETH as the initial balance in your account.
* To add dummy Ethers, click on the Deposit and Get Ether button under Test Faucet.

**4.Use Editor Remix to Write the Smart Contract in Solidity**

* We will use Remix Browser IDE to write our Solidity code. Remix is the best option for writing smart contracts, as it comes with a handful of features and offers comprehensive development experience.
* It is usually used for writing smaller sized contracts. Remix’s features include:
* Warnings like Gas cost, unsafe code, checks for overlapping variable names, and whether functions can be constant or not.
* Syntax and error highlighting.
* Functions with injected Web3 objects.
* Static analysis.
* Integrated debugger.
* Integrated testing and deployment environment.

5.**Create a .sol Extension File**

Open Remix Browser, and click on the plus icon on the top left side next to the browser to create a .sol extension file.

**6.Deploy Your Contract**

* Deploy the smart contract at the Ethereum test network by pressing the deploy button at the right-hand side of the Remix window. Wait until the transaction is complete.
* To check the tokens in your wallet, go to the main window, click add tokens, enter the smart contract address, and click ok. You should be able to see the number of tokens there

**Steps to Deploy Ethereum Smart Contracts**

To make your smart contract live, switch to the main Ethereum network at CryptoShare.App:

* Add some real Ethers.
* Now, again deploy your smart contract using remix, as mentioned in the above steps.
* When a smart contract is deployed successfully, visit http://www.etherscan.io and search your smart contract address there. Select your smart contract.
* Now you need to verify your smart contract here, click verify contract.
* Copy your smart contract code and paste it at Etherscan. Select the same compiler version that you selected at remix to compile your code.
* Check optimization to Yes if you had selected optimization at remix; otherwise, select No.
* Click Verify.
* It will take a few minutes, and your smart contract will be live if no issues occurs

## **ETHEREUM BLOCKCHAIN LIMITATIONS**

**IMMUTABILITY**

* Smart contracts in Ethereum are immutable by default. Once you create them there is no way to alter them, effectively acting as an unbreakable contract among participants.
* However, for some scenarios, it is desirable to be able to modify them. Think of a traditional contract between two parties: if they both agreed to change it, they would be able to do so. On Ethereum, they may desire to alter a smart contract to fix a bug they found (which might even lead to a hacker stealing their funds!), to add additional features, or simply to change the rules enforced by it.

## **SCALABILITY**

## The lack of scalability has always been one of the disadvantages of blockchain technology. On Ethereum, it is probably the most talked-about problem the blockchain network faces, mostly because it is more evident than others. Through the ICO boom and DeFi summer, transaction volumes on Ethereum surged astronomically, as expected with widespread adoption. One small problem, however, arguably stopped further expansion: the network’s speed, or lack thereof.

* Ethereum today can only process about 15-30 transactions per second. With surging transaction volumes, slow TPS figures will lead to network congestion, resulting in delayed transaction finality and humongous transaction fees.

**INCREASING TRANSACTION COSTS**

* Critics claim that Ethereum's gas fees should be around $0.05, ideally. During the DeFi boom and the gas wars that followed, average gas fees on Ethereum soared to around $60-$70. Within the Ethereum ecosystem. gas fees are simply bids for including a set of transactions in the limited block space. In case of high demand, these bids go up as users attempt to fast-track their transactions to gain an edge. This results in extremely high transaction costs.
* At such levels, interacting with applications on Ethereum is beyond the reach of the average Web3 user, making developers opt for cheaper blockchain networks like Solana and BSC.
* It is important to note that the Merge has not solved Ethereum's gas fees challenges. However, recent upgrades and the crypto market mood have seen average gas prices drop from around $40 to $15.55 as of the second half of 2022.

**CENTRALISATION FEARS**

* + When Ethereum existed as a proof-of-work blockchain, some mining pools were considerably larger than others and, as such, had more consensus power. Of course, this model sparked talks of centralization, which defeats everything Ethereum represents: a secure, scalable, and decentralized blockchain network.
  + Today, Ethereum has completed its transition (known as The Merge) to a proof-of-stake model after launching a Beacon chain in 2021 in preparation. This model, some argue, reduces the possibility of a 51% attack because it makes an attack more costly. The new arrangement has phased out the miners, replacing them with validators who staked Ether (ETH).
  + More centralization fears have emerged as institutional capital and large exchanges vie for a large chunk of ETH staking power.

**PRIVACY**

* Privacy is almost non-existent on the network and many other Layer-1 blockchains, which presents a problem. Ethereum is a fully transparent blockchain, which means that every interaction is not only stored on the ledger but accessible to the public. While transparency may be a good thing, its extreme form turns it into one of the Ethereum blockchain’s major disadvantages, especially for institutional players and DeFi traders.
* Zero-knowledge rollups built on Ethereum have solved the privacy challenge to some extent, but these solutions cannot pass off as traditional blockchains. Many of them are limited and unable to support smart contract deployment. For example, zkEVMs are still in a work-in-progress stage, and without EVM compatibility it would be impossible for Layer-2 privacy solutions to reach mass adoption.

**LAYER 2 SOLUTIONS**

* As the number of users increases and the number of simultaneous transactions goes up, a Layer 1 blockchain can become slow and expensive to use. This is especially true of Layer 1 blockchains which use a Proof of Work mechanism as opposed to Proof of Stake.
* Layer 2 solutions handle transactions off the Ethereum Mainnet to achieve scalability.Below are some of the most commonly used Layer 2 solutions:

**1.Polygon (MATIC)**

* By far, Polygon is the most widely adopted layer 2 solution for Ethereum. Unlike Ethereum, which is limited to 13–17 transactions per second (TPS), Polygon can execute up to 7,000 TPS, making it comparable to Visa.
* As a proof-of-stake (PoS) network—to which Ethereum is transitioning soon—Polygon relies on MATIC tokens to verify transactions. Therefore, those who hold MATIC tokens can become the network's validators, earning a cut whenever a transaction occurs. This process is called staking.
* Likewise, MATIC holders can delegate their MATIC stash to trusted validators, serving the same purpose but indirectly. That's why they are called delegators. Either way, staking MATIC yields up to a 9.5% annual percentage rate (APR). This is far beyond the national average offered by banking savings accounts at 0.06%. Polygon's calculator shows exactly how much one can earn with MATIC staking.

**2.Arbitrum**

* Arbitrum has gained much popularity in a very short time since its launch in May 2021. At this pace, it may even outcompete Polygon with the current TVL at $3.3 billion. Interestingly, the company's founder that developed Arbitrum, Offchain Labs, is none other than former White House Deputy Chief Technology Officer, Ed Felton.
* Unlike Polygon, Arbitrum doesn't have its own token. Therefore, it doesn't have a staking mechanic. Instead, Arbitrum uses Ethereum's main chain to verify transactions. For this reason, Arbitrum's gas fees are somewhat higher than on Polygon but still significantly lower than on Ethereum.
* Yet, much is expected of Arbitrum because it has its own Arbitrum Virtual Machine. This is the framework powering its smart contracts, just like EVM (Ethereum Virtual Machine). At first glance, this may seem like a problem, but it is a benefit because it doesn't rely on Ethereum if it significantly changes its consensus protocol.

**3.Loopring**

* Loopring uses zero-knowledge (ZK) rollups to stand out from the L2 herd. Rollups just mean that L2 networks scoop up Ethereum's main chain (L1) data and feed it back in a compacted format. Both Arbitrum and Optimism rely on optimistic rollups to accomplish this. However, as their name implies, optimistic rollups count on all network participants to act in good faith.
* Loopring takes a different approach with ZK rollups. Without going into cryptography minutia, this means that transactions are verified without other parties revealing their identity. The ZK approach also results in greater data throughput because its type of rollup significantly reduces transaction volume.
* Another ZK benefit is that there is no lengthy challenge period because transaction validity is inherent. Therefore, withdrawal time is faster than both Polygon and Arbitrum. With these upsides, Loopring also has its own stakeable token called LRC

**4.Immutable X**

* Founded by young Australian brothers Robbie and James Ferguson, Immutable X has become the go-to L2 network for NFT marketplaces. As the first sidechain specialized in NFT trading, Immutable X doesn't charge any gas fees and provides an amazing transaction speed of up to 9,000 TPS with instant confirmation.
* For these reasons, Immutable X has been stacking a string of major partnerships recently. Iconic gaming retail chain GameStop will use the L2 protocol for its upcoming NFT marketplace. Likewise, Immutable X also powers TikTok's Top Moments NFTs. It also runs Illuvium, Planet Quest, Guild of Guardians, Gods Unchained, Highrise, GreenPark Sports, and ESL Gaming in the blockchain gaming sphere.
* But how does Immutable X achieve this amazing performance at zero fees, making it so attractive?
* In short, Immutable X partnered with StarkWare, a company specializing in upscaling blockchain without compromising security. Specifically, they have developed StarkEX, a validity-proof rollup, and StarkNet, a permissionless decentralized ZK-rollup. Validity-based proofing means that crypto/NFT assets cannot be traded without the user's specific permission.

## **MODEL FOR AN ETHEREUM SMART CONTRACT**

Solidity is a high-level Javascript-like language developed specifically for creating smart contracts. It is typed statically and supports libraries, inheritance and complex user-defined types.

Solidity compiler converts code into EVM bytecode which is sent to the Ethereum network as a deployment transaction.

Therefore this fruit model will be implemented in Solidity for it is influenced by high-level languages such as Python,JavaScript,C++.Below is a step by step guide of how the model code is executed.

1.Create an array to hold the names of fruits

2.Create a function to push a new value to the array

3.Create a function to change the value at given index

4.Create a function to delete a value

5.Create a function to return the array

Now we are implementing our smart contract in Solidity:

* **Create a new file in contracts directory and call it Fruits.sol. We’ll start the file by indicating the license and solidity version we are supporting:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.4.21 <0.7.0;

* **Next, declare the contract scope where we are going to write all the code:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.4.21 <0.7.0;

contract Fruits {

}

* **Create an array to hold the fruits:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.4.21 <0.7.0;

contract Fruits {

string[] myFruits;

}

This is a string array with a private modifier, which means it can’t be accessed outside the contract and thus we can’t change the value directly.

* **Next, create a function to add a new value**:

// SPDX-License-Identifier: MIT

pragma solidity >=0.4.21 <0.7.0;

contract Fruits {

string[] myFruits;

function addFruit(string memory fruitName) public {

myFruits.push(fruitName);

}

}

We created a function called addFruit, which accepts a string as a parameter, fruitName. This is declared public so it can be called by the UI or terminal. In the function body, we are simply pushing the value to the array

* **Update the value:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.4.21 <0.7.0;

contract Fruits {

string[] myFruits;

function addFruit(string memory fruitName) public {

myFruits.push(fruitName);

}

function updateFruit(uint fruitIndex, string memory newFruitName) public returns (bool) {

if(myFruits.length > fruitIndex){

myFruits[fruitIndex] = newFruitName;

return true;

}

return false;

}

}

updateFruit accepts two arguments, fruitIndex and newFruitName, and returns a boolean value. It works like this: if the index is out of the bounds of the array, it returns false. Otherwise, it changes the value of the array with a new provided fruit name at the provided index and return true.

* **The next steps is to create the delete function:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.4.21 <0.7.0;

contract Fruits {

string[] myFruits;

function addFruit(string memory fruitName) public {

myFruits.push(fruitName);

}

function updateFruit(uint fruitIndex, string memory newFruitName) public returns (bool) {

if(myFruits.length > fruitIndex){

myFruits[fruitIndex] = newFruitName;

return true;

}

return false;

}

function deleteFruit(uint fruitIndex) public returns (bool) {

if(myFruits.length > fruitIndex){

for(uint i=fruitIndex; i < myFruits.length-1; i++){

myFruits[i] = myFruits[i+1];

}

myFruits.pop();

return true;

}

return false;

}

}

Here we are checking the index out of bounds condition and then updating the array by replacing the value with the next value from the provided index. This way, the value at the provided index will be lost. In the end, we pop out the last value and return true.

* **The last step is to return the array.**

To read all the values of the array:

// SPDX-License-Identifier: MIT

pragma solidity >=0.4.21 <0.7.0;

contract Fruits {

string[] myFruits;

function addFruit(string memory fruitName) public {

myFruits.push(fruitName);

}

function updateFruit(uint fruitIndex, string memory newFruitName) public returns (bool) {

if(myFruits.length > fruitIndex){

myFruits[fruitIndex] = newFruitName;

return true;

}

return false;

}

function deleteFruit(uint fruitIndex) public returns (bool) {

if(myFruits.length > fruitIndex){

for(uint i=fruitIndex; i < myFruits.length-1; i++){

myFruits[i] = myFruits[i+1];

}

myFruits.pop();

return true;

}

return false;

}

function getFruits() public view returns (string[] memory) {

return myFruits;

}

}

**OTHER PROGRAMING LANGUAGES THAT SUPPORT ETHEREUM SMART CONTRACTS**

**JAVASCRIPT**

JavaScript is a general-purpose programming language, and it’s found a place in the blockchain space. Because JavaScript is an entry-level language, most blockchains tend to create a JavaScript wrapper or library to allow developers to easily jump into the ecosystem and start building amazing products as soon as possible.

Hyperledger Fabric is a blockchain that allows you to build a smart contract with a few programming languages, including JavaScript (Node.js).

The community built web3.js, a collection of libraries that allows you to interact with Ethereum smart contracts using HTTP, WebSocket, or IPC.

**RUST**

Rust is a low-level statically-typed programming language that is fast and memory-efficient — in an industry where scalability is not negotiable, Rust, as a language, finds a home. Rust is a relatively new programming language with enormous power while retaining simplicity, memory efficiency, reliability, and complexity combined.

Rust contains some object-oriented features; you can create structs and data. But unlike other object-oriented languages, it does not exhibit inheritance.

Smart contracts blockchain using Rust include Solana, Polkadot, and Near Blockchain. You can find many blockchain projects built with Rust in this GitHub repository.

**YUL**

Yul is an intermediate programming language that is compiled to bytecode for addressing the needs of different backends. The Solidity compiler has an experimental implementation that uses Yul as an intermediate language. Yul is used in stand-alone mode and for inline assembly inside Solidity.

Yul bears planned support for EVM and ewasm (Ethereum flavored WebAssembly). It is designed to be a usable common denominator of both platforms.

Yul is a great target for high-level optimization stages that can benefit both EVM and ewasm platforms equally