**Development phase**:

In this Decentralized application, we're using Solidity, [React.js](https://reactjs.org/), CSS, HTML  to build our project. We need a way to manage all of the client-side behavior of our application, and store data from the blockchain. Instead of doing this all by hand, react is an open-source, front end, JavaScript library for building user interfaces or UI components. It gives us this ability out of the box by organizing our code into reusable components and managing our application state with its state object. React helps us to create application which is connected with blockchain smart contract. It has Declarative views make your code more predictable and easier to debug

Examples of React uses can be found here:

<https://reactjs.org/community/examples.html>

**We'll create:**

* Ethereum smart contracts with the Solidity programming language
* We’ll write tests for the smart contracts in JavaScript
* We’ll deploy to the smart contracts to a blockchain
* We’ll create a client-side website with Web3.js and React.js so that users can talk to the smart contracts.

Here, we will create smart contract and test cases which check the smart contract is working properly. This smart contract is deployed to a blockchain. There are different ways to deploy. After this, client-side user interface is created with above technologies.

Normally when you interact with a web application, you use a web browser to connect to a central server over a network. All the code of this web application lives on this central server, and all the data lives in a central database. Anytime you transact with your application, must communicate with this central server on the web.

Here, in the implementation we have two parties i.e. Buyer and seller parties. Seller can post the item on the marketplace and seller can also become buyer to purchase the item. Buyer can purchase the item, which is present on the marketplace.

**So, what is marketplace?**

It is a place where products are displayed, and user can purchase anything present on this website (i.e. Marketplace). According to Wikipedia, an **online marketplace** is a type of e-commerce website where product or service information is provided by multiple third parties. In our case, third party can be seller as well as buyer.

Workflow of Buyer and seller is as follows:

Diagram

Description automatically generated

**Buyer of Product:**

Whenever buyer clicks on the buy option to purchase, they will submit the id of the product that they want to purchase (this will be handled by our client-side application). Additionally, they will send Ethereum cryptocurrency from their wallet to purchase the product when they call this function.

This buyer’s function is made payable, which means that it will accept Ethereum cryptocurrency. We can see this in action momentarily because we want to pay the owner, we must update the existing struct and event to use the address payable.

This test works much like the sellProduct() function test. It does a few key things:

Checks that the product was transferred to the buyer

Checks that the seller received the Ether cryptocurrency funds automatically

Checks all failure cases to make sure that we protect against them inside the function

**Seller of Product:**

We'll create the first feature, which will allow a user to list an item for sale in the marketplace. In order to do that we'll need to model the product.

Solidity allows you to create your own data structures, with any arbitrary attributes. That's exactly what we've done by creating a Product struct. It stores all the attributes of a product that we'll need, like id, name, price, owner, and purchased. This is stored same as associative array or hash.

Text

Description automatically generated

Next, we need a place to store this product on the blockchain. We'll create a mapping on Solidity like this:

Mappings work like associative arrays, or hash tables, with key value-pairs. Mappings have unique keys that return unique values. In our case, we will use an id as a key, and the value will be a Product struct. This will essentially allow us to look up a product by id, like a database.

**However, blockchain is decentralized and hence there is no central place for it to be stored. That's why it is stored in computers or systems all across the network. These systems or computers are known as nodes. Each of the nodes has one copy of the blockchain or in other words, the transactions that are done on the network.**

Next, we want to keep track of how many products exist in the smart contract with a productCount counter cache like this:

**This function will do a few things:**

Create a new product with a struct

Add the struct to the mapping, and store it on the blockchain

Trigger an event that lets someone know a product was created.

We can create that function like this:

First, we create a function that accepts

Name and price arguments. The product price is expressed in Wei, Ether's smallest subdivision.

Next, we add some requirements that must be satisfied before the function continues execution. We check that the name is present, and that the price is greater than 0.

Next, we generate a new product id by increasing the product count with the ++ operator (this just adds 1 to the previous value).

Then we create a new product and add it to the mapping. Note that msg.sender is the address of the user creating the product.

Finally, we trigger an event to let everyone know that the product was created successfully.

This function helps to create and keep the count of total products.

Text

Description automatically generated

Now, coming to building the client-side application for the Marketplace. Here's what we'll do in this section:

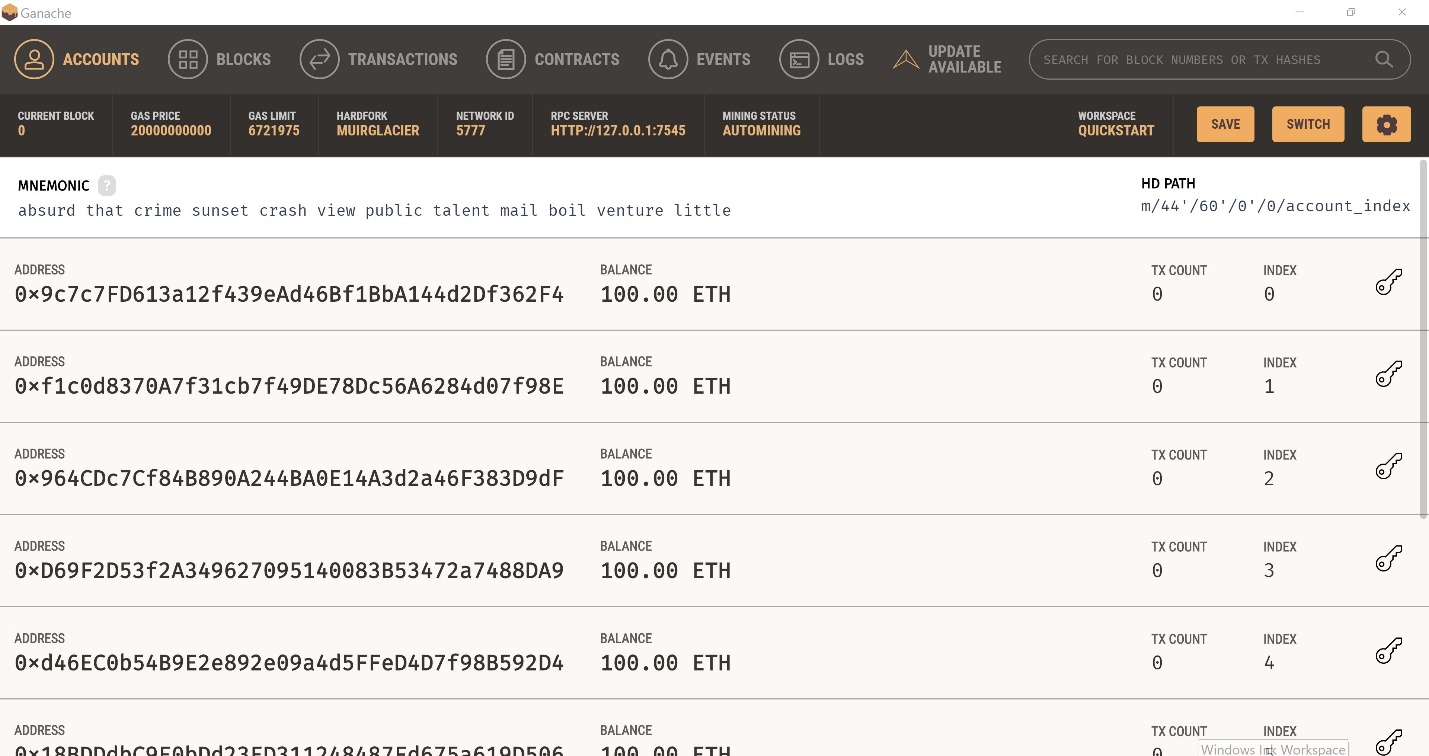
* Start the app, and run the starter kit in our browser
* Connect our web browser to the blockchain
* Connect our web app to the blockchain and start talking to the Marketplace smart contract.

we need to build our application:

* React.js for building out the interface
* Bootstrap for creating UI elements without writing CSS
* Web3.js for connecting our app to the blockchain

Now let's connect our web browser to the blockchain. We'll use Metamask for this. To do this we'll need to do two things:

* Connect Metamask to our Ganache personal blockchain instance.



Here, we have some terms as you can see in this ganache application. Those are:

**Accounts**: It shows list of account available for use. It consists of private key, public key and balance. If there is need to import the account to metamask or any other extension, we need to use the private key.

**gasPrice**: Sets the default gas price for transactions if not otherwise specified. Must be specified as a hex encoded string in wei. Defaults to (2 gwei).

**gasLimit**: number Sets the block gas limit. Must be specified as a hex string or number(integer).

**network\_id**: Specify the network id ganache-core will use to identify itself (defaults to the current time or the network id of the forked blockchain if configured)

**mnemonic**: Use a specific HD wallet mnemonic to generate initial addresses.

* Import some accounts from Ganache into Metamask so that we can act on their behalf as users of our marketplace application

Graphical user interface, application

Description automatically generated

Graphical user interface, application

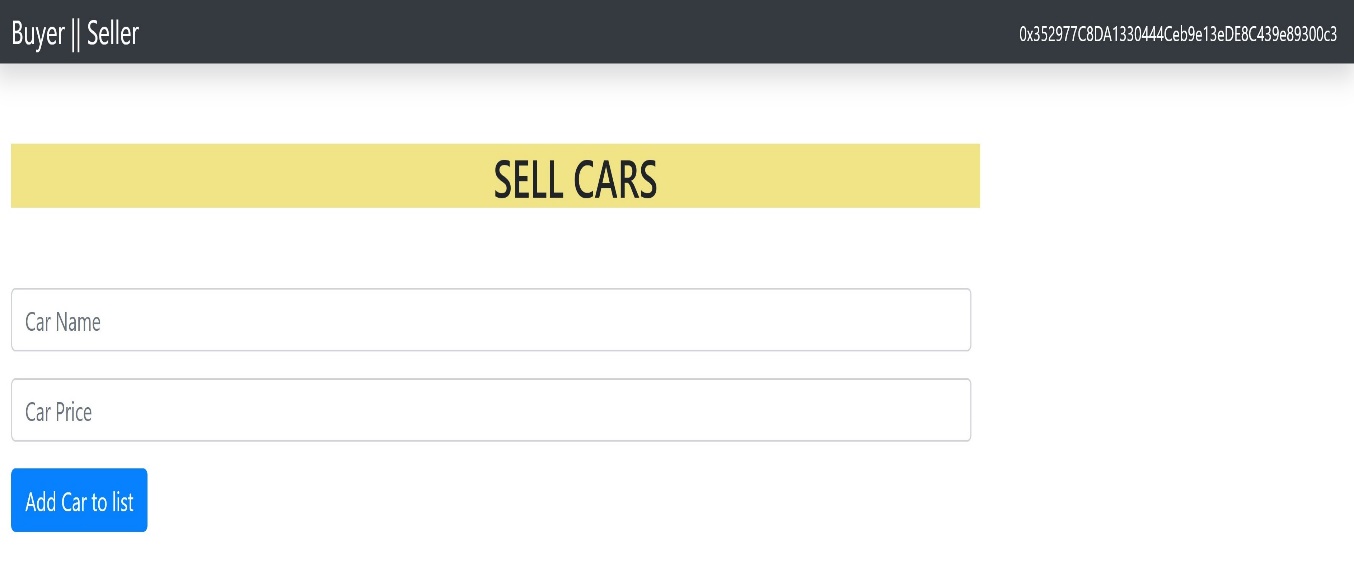
Description automatically generatedGraphical user interface, application

Description automatically generated

There is inclusion of a function which detects the presence of an Ethereum provider in the web browser, which allows us to connect our decentralized application to the blockchain. Multiple accounts can be used to perform the transaction.

**Seller Front-End**:

* Create a react component that holds the data for our code, including a form that allows users to list new products, and a table that shows products for sale.
* We'll wire up the form so that users can actually list their product for sale on the blockchain.

****

**Buyer Front-End:**

Now let's finish off our app by allowing users to buy products in the marketplace. We'll do two things in this section:

* List all of the products onto the page
* Allow users to buy them with the click of a button

**Graphical user interface, application

Description automatically generated**

**Manual Testing:**

In this phase, smart contract should be tested on test-net.

There are 2 process to access the test network:

1. Using command line : Test network can be accessed through terminal, which is complicated way and requires commands
2. Using Extensions like Metamask.

The smart contract can be tested using command line using libraries like geth, truffle. Once the geth is installed you can use testnet like (Rinkeby/Ropsten/kovan) through cmd.

The smart contract can be tested using extensions like metamask as well. There is a need to download metamask from their official website and add to browser extension.

All the test-net transaction and state changes requires facets that is like ethers used on main-net. For example: Facets are the currency used in the world of blockchain. There may be possibility of deployment failure of smart contract because of various reasons. One of many is insufficient gas/ ethers available at your address. To interact with smart contract using MetaMask, you’ll need Ether in your wallet. You can add through Get Ether option from buy in metamask, a faucet for the Rinkeby or any specific test-net. You can add ethers into your account using direct deposit option as well. If the smart contract is still failing, then there is need to go back to the development stage.

**If we were to build our marketplace application on the web, we’d run into a few problems:**

1. The data on the database could be changed: it could be counted more than once or removed entirely.
2. The source code on the web server could also be changed at any time.

This test does two things:

* Checks that the smart contract has an address, i.e., it was successfully deployed to the network.
* Checks that the name was set when it was deployed.

**The smart contract is compiled and deployed, as you can see the details of network used, cost related to deployment.**

Text

Description automatically generated

Text

Description automatically generated

**Testing and deployment:**

In the final stage, once everything is compiled and set, we can deploy it using test-net or main-net. All the Smart contracts ready for production should be audited before deploying on main net, because even though business logic of smart contracts is tested on test-net several times, smart contract cannot be declared as secured or bug free contract, smart contract may contain some logical errors that can be identified.

**We want to ensure that the contracts are bug free for a few reasons:**

1. All of the code on the Ethereum blockchain is immutable; it cannot change. If the contract contains any bugs, we must disable it and deploy a new copy. This new copy will not have the same state as the old contract, and it will have a different address.

2. Deploying contracts costs gas because it creates a transaction and writes data to the blockchain. This costs Ether, and we want to minimize the amount of Ether we ever have to pay.

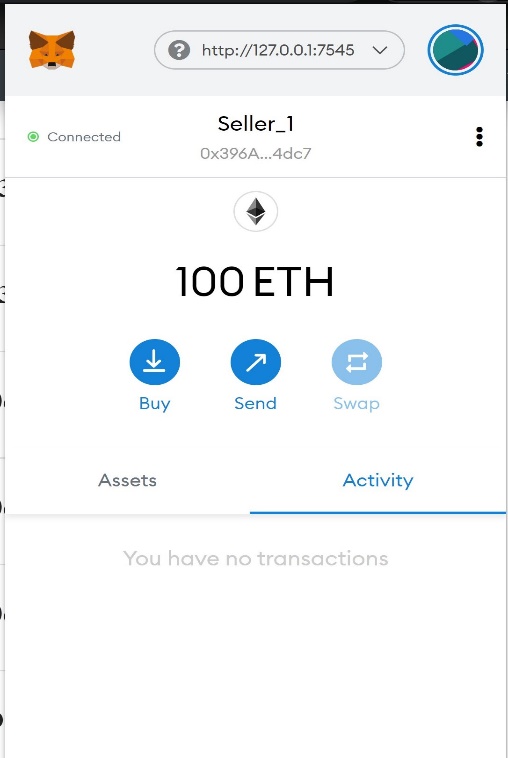
3. If any of our contract functions that write to the blockchain contain bugs, the account who is calling this function could potentially waste Ether, and it might not behave the way they expect.

After deployment, every function execution requires ether i.e. every transaction after deployment requires ethers.

**For example, consider function as task when it is completed user has to pay some cost in form of ethers.** A blockchain is a globally shared, transactional database. This means that everyone can read entries in the database just by participating in the network. Each transaction is provided with transaction hash and block number, they help to keep track of transactions and provide secure transaction between two parties.

* First, we fetch the product from the mappings and also allow seller to post and create a new copy of it in memory.

Initially, **seller and buyer account have 100 ethers each**, when seller post the advertise, 0.003536 ether are deduced from the sellers account (i.e. the processing fee).

 Graphical user interface, application

Description automatically generated

* Then we store the current owner to a variable. We will transfer the ownership of the product inside this function, so we need to know who the original owner was. For example, owner details are included with the car details (i.e. **public address of seller and once the product is purchased the public address of buyer is displayed**).



* Next, we add a few requirements. We check for a valid id, that there is enough Ethereum cryptocurrency in the transaction, that the buyer is not the seller, and that the product has not been purchased already.

Make sure the seller and buyer are different because the ownership of vintage car is changed accordingly.

* Then we facilitate the transaction. We transfer the ownership to the buyer, mark the product as purchased, and add the product back to the mapping. After this, we send the cryptocurrency payment to the seller.

**Suppose** the buyer wants to buy the Shelby cobra which costs 50 Ethers, buyer must have enough ether to buy the car. Once it is verified, Buyer sends request to buy and validation is done.

* 1. When buyer request to buy car, buyer has to confirm the request and request include the gas fee (i.e. the fee required to perform the transaction and actual cost of car)

Graphical user interface, application

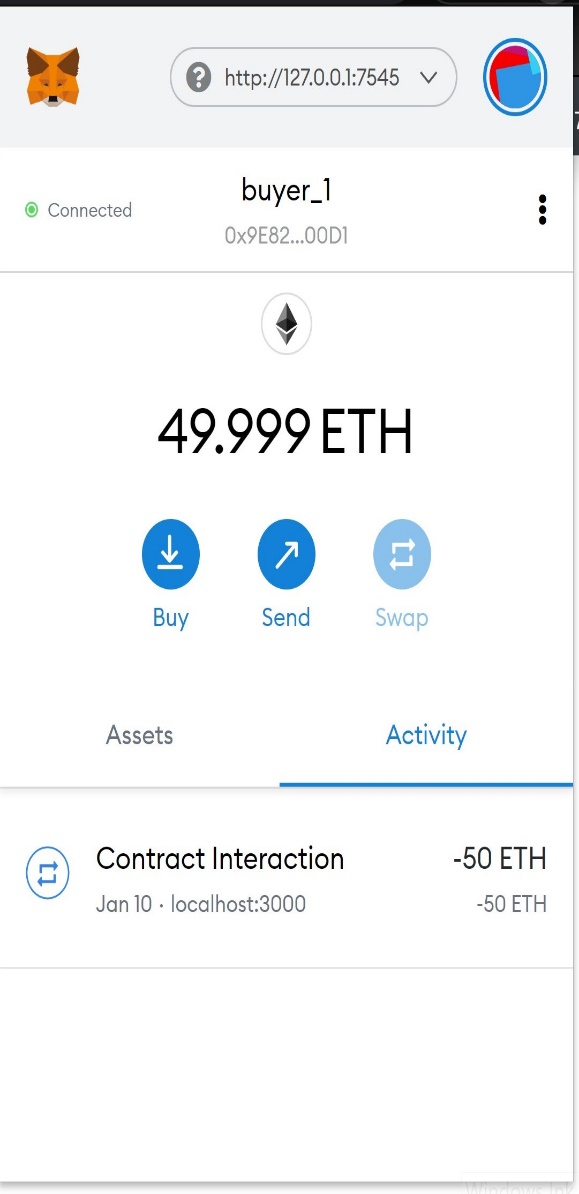
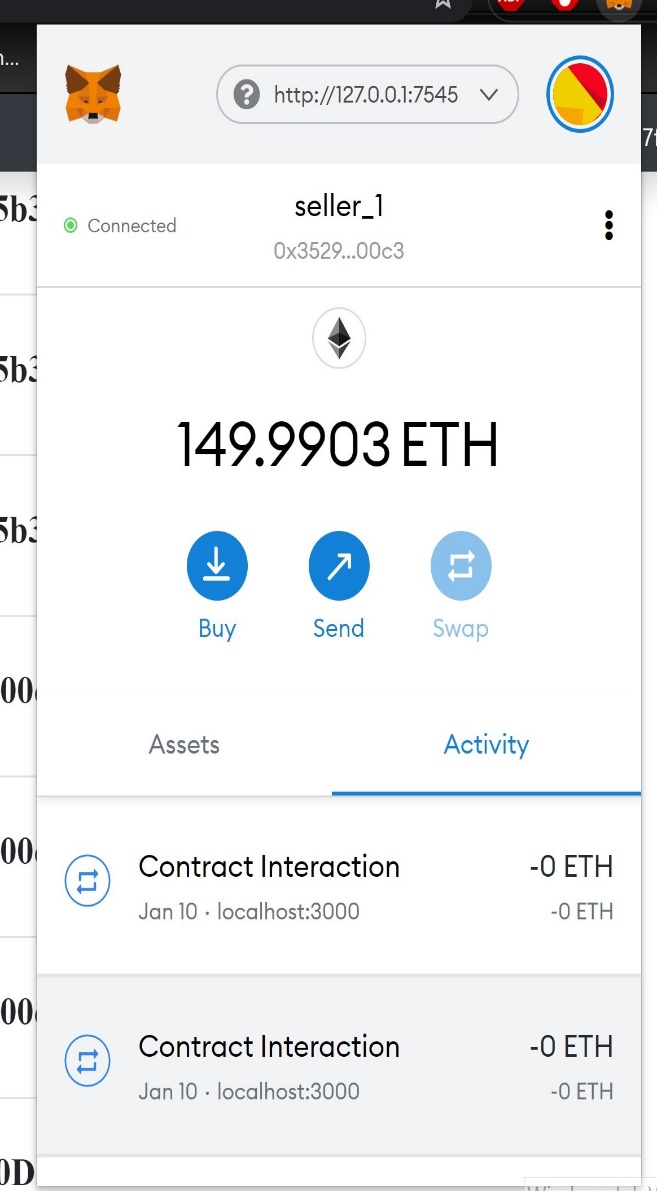
Description automatically generated

* 1. After the confirmation from buyer side, contract interaction details are obtained which includes public address of smart contract and user, total amount, gas limit and gas used, time of transaction and total cost.

Graphical user interface, text, application, email

Description automatically generated

* 1. After the transaction, seller account is transferred the amount of purchase (i.e. 50 Ether) and buyers account is updated as well.

**Smart contracts are better than non-smart ones?**

Everyone knows what smart contract is and how it works. But the important question is how it is better.

Every smart contract run on the blockchain, so they are stored on a public node and cannot be changed. The transactions that happen in a smart contract are **processed by the blockchain**, which means that there is no other third. **This means there is no one to rely on!** The transactions only happen **when the conditions in the agreement are met** there is no third party, so there are no issues with trust.

For example:

Consider there is Mark, who is seller post the advertise to sell vintage car on the marketplace, and john sees the advertise of Aston martin DB9 for 100 ethers and he wants to buy it. Smart contract consists of agreement between two users which helps in smooth transaction and processing. When john purchase it, seller gets the total amount of car and ethers are deduced from johns account instantly.

Whereas, if john wants to purchase a vintage car, first he has to search for the car, there are other third-party companies which he has to pay to purchase one.

**Benefits of smart contract?**

* **Smart contact helps to precise and accurate:**

Smart contract which is working must be accurate and record all terms and conditions in explicit detail. This is a requirement because an omission could result in transaction errors.

* **Smart contract is transparent:**

The terms and conditions of these contracts are fully visible and accessible to all relevant parties. There is no way to dispute them once the contract is established. This facilitates total transparency of the transaction to all concerned parties.

* **Transactions are fast and efficient:**

All the transactions are fast and executed efficiently if there is no problem or bugs in the smart contract.

* **PEER-TO-PEER Transactions:**

Smart contracts can be used for a whole range of peer-to-peer transactions. Users of all shapes and sizes can use these platforms to create and agree on smart contracts. These contracts then remain active until a set of agreed conditions are met. Once the smart contract is happy that all conditions have been met, it then allows the remaining portion of the agreement to be fulfilled.

So far, smart contracts have been used for everything from launching new products to selling goods on the internet. Companies are also using smart contracts to secure the services of development teams and other outsources companies.

* **Transactions are secured:**

All the transactions are provided with security. **Let’s explain security in Ethereum in detail.**

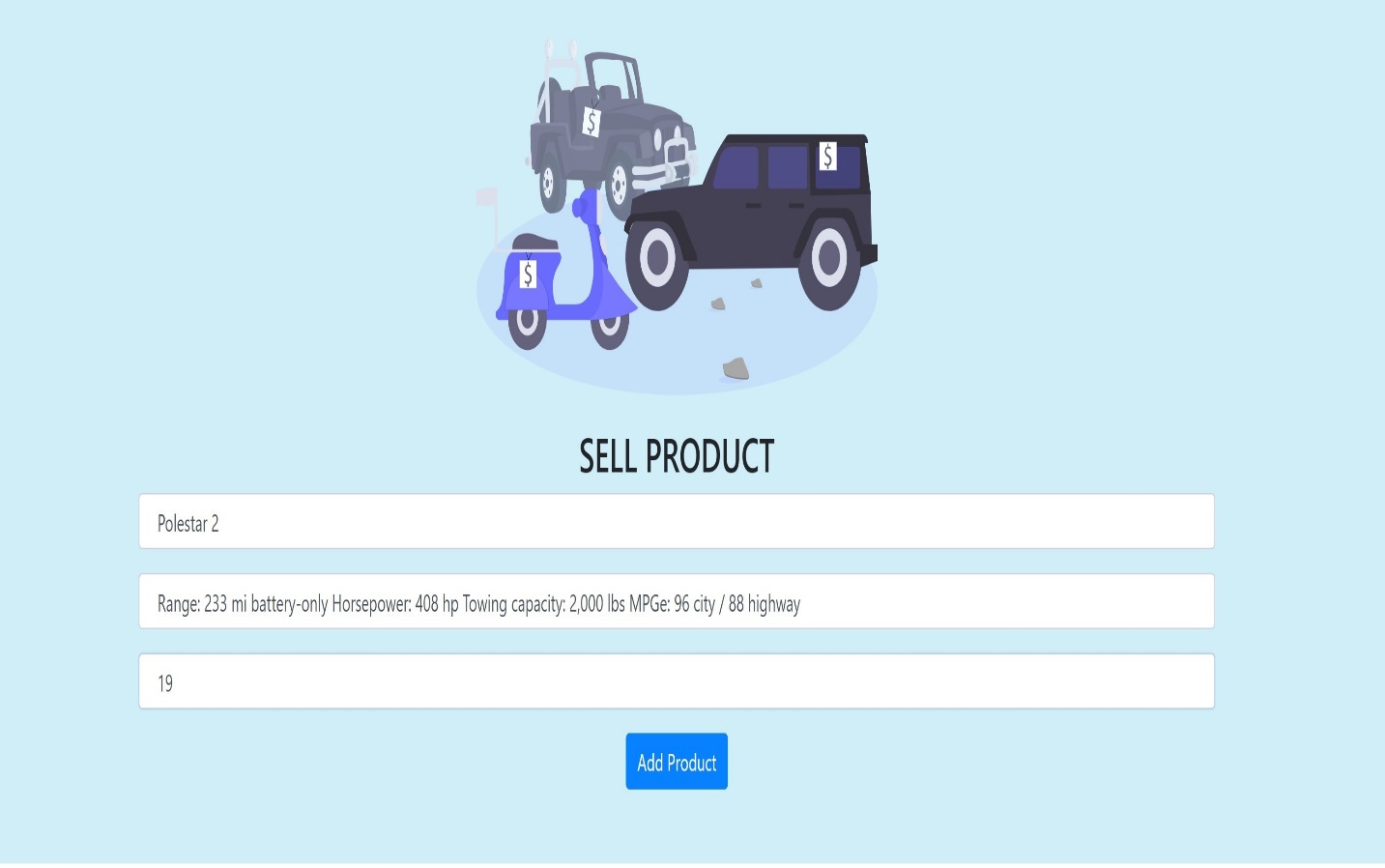
* **All data is paper-less:**

**Table

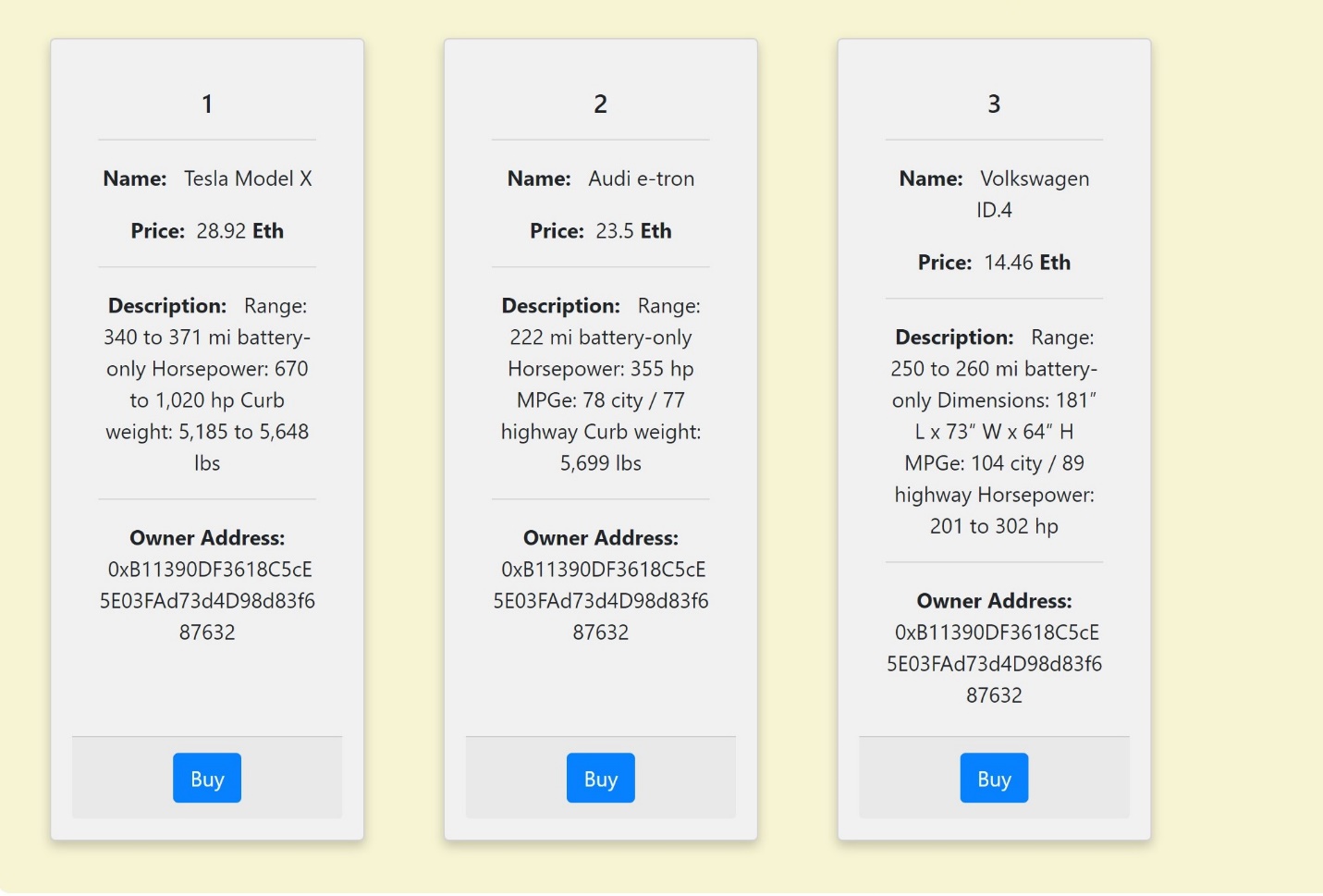
Description automatically generated**

**UPATES:**

Changes are made in the smart contract which includes feature to include Product description. For example, consider the seller wants to sell product and wants to add description about product as-well. Seller can add it easily through this input description. User wants to sell car so he/she can add in description about car and add into listing so buyer can purchase it.



Buyers can purchase the product, if one has enough ethers in the account and when purchased the buy changes to sold.



**Security in Ethereum Smart contract:**

Ethereum uses the protocol called as **proof -of -work** which helps in providing the protection from the cyber-attacks, service attacks, etc. There is little incentive for a subset of miners to start their own chain. The objective of **proof -of -work** is to extend the chain. The longest chain is most believable as the valid one because it's had the most computational work done on it. Within Ethereum's proof -of -work system it's nearly impossible to create new blocks that erase transactions or create fake ones or maintain a second chain. That's because a malicious miner would need to always be solving the block nonce faster than everyone else.

To consistently create malicious, yet valid, blocks, you'd need over 52% of the network mining power to beat everyone else. You'd need a lot of computing power to be able to do this amount of "work". And the energy spend might even outweigh the gains you'd make in an attack. Proof- of -work make use of hashing which helps in generating values or values from the string of text or data from users using mathematical functions.

Hashing not only plays role in providing security in smart contract but also in data protection, password protection, etc. Hashing is vast term but in simple it converts the main data into ascii values mixed with string, numbers. This makes it difficult for hackers to track the transaction or data. There are different algorithms used to provide protection.

For example, consider user sends transaction request and ethers are deduced from user’s account. It is the time, when data is venerable but with hashing, it minimizes the chances of tampering of data. One of the ways, hackers hack the system is collecting the ascii data of transactions and comparing those with different probabilities. However, now the hashing algorithms are powerful as well, when user send ethers to different accounts, a special hash key is generated and it is attached with transaction details, when this transaction arrives at other end it compares the hash key if they are similar then a secure transmission has occurred. It also ensures that unauthorized users can’t alter the original data.

**How can we know that smart contract is working?**

The core use for a smart contract would be the automation of some specific processes or task, which requires a middleman. With the help of blockchain we can design smart contract to remove the traditional processing of contract. Because of blockchain technology, we are able to [decentralize](https://en.wikipedia.org/wiki/Decentralization) smart contracts so that they are work. By decentralizing, I mean that they are not controlled by one central party. The blockchain is a shared database run by many computers (called 'nodes') belonging to many different people. Because of this, not one single person or company has control of it.

When the smart contract is deployed, the purpose of smart contract can be checked by the functions defined in the smart contract. For example, in our case, when seller post the advertise on the page and buyer can purchase vintage car using the ethers which he has, and it is executed smoothly. **If all the information in the start of document is working and transactions are performed, we can say that smart contract is working properly.**

**Similar Platforms like Ethereum which has Smart contracts:**

**There are many platforms which has smart contract use, here are some of them,**

**EOS:**

An EOSIO Smart Contract is software registered on the blockchain and executed on EOSIO nodes, that implements the semantics of a “contract” whose ledger of action requests are being stored on the blockchain.

EOS uses c++ to [develop smart contracts](https://blockgeeks.com/guides/smart-contract-development/).

The reasons why EOS chose WASM are as follows:

* **Flexibility**: Developers can code in the language of their choice.
* **Speed and Efficiency**: It executes at native speed by taking advantage of common hardware capabilities available on a wide range of platforms.
* **Open and Debuggable**: It is designed to be pretty-printed in a textual format for debugging, testing, experimenting, optimizing, learning, teaching, and writing programs by hand.
* **Safe:** It describes a memory-safe, sandboxed execution environment that may even be implemented inside

**Cardano:**

[Cardano’s](https://blockgeeks.com/guides/what-is-cardano/) approach is unique in the space itself since it is built on scientific philosophy and peer-reviewed academic research.

The Cardano team wants to adhere to a set of principles and philosophies. They did not set out with a proper roadmap or a [white paper.](https://blockgeeks.com/guides/crypto-whitepaper/) Instead, they focused on embracing a “collection of design principles, engineering best practices, and avenues for exploration.

The following are these principles, and they are taken directly from the [Cardano](https://blockgeeks.com/guides/what-is-cardano/) website.

* Separation of accounting and computation into different layers.
* Implementation of core components in highly modular functional code.
* Small groups of academics and developers competing with peer-reviewed research.
* Heavy use of interdisciplinary teams including the early use of InfoSec experts
* Fast iteration between white papers, implementation and new research required to correct issues discovered during the review.
* Building in the ability to upgrade post-deployed systems without destroying the network
* Development of a decentralized funding mechanism for future work
* A long-term view on improving the design of cryptocurrencies so they can work on mobile devices with a reasonable and secure user experience.

**WHAT ARE THE BENEFITS OF SMART CONTRACTS?**

Smart Contracts bring many benefits to the processes that will make a difference with those in which old-fashioned techniques are used.

**Efficient execution**:

The processes that are most likely to be implemented are the ones that are currently manual processes, so the increase of speed will be evident.

**Security:**

Smart Contracts make use of encryption at the Blockchain level. This kind of technology is the most secure available.

**Lower cost:**

There are no intermediaries or cost fees during the execution process.

**Accuracy:**

There is no human intervention during the execution process. The whole transaction is less error prone.

**No execution risk:**

The process runs in a decentralized architecture supported by the network nodes rather than a centralized party. The autonomous smart contracts executed way faster compared to the old-fashioned traditional approach. As all the parameters are already defined within the smart contracts, it only needs to match them before it starts executing.

**Autonomous:**

The biggest benefit of smart contracts is the automation that it offers. In simple terms, it means that it is interruption-free, and no third party can make changes in the agreement and decision. This automation can go a long way as it helps organizations to automate certain aspects of their business. Not only that, it resolves issues in some processes where trust is an issue.

**Trustless:**

The whole system is trustless. This means that there is no need to trust other parties. Sounds counter-intuitive? Well, in simple words, it means that there is no need to trust the parties to carry out a transaction. A transaction or a trade does not require trust as its integral part. As smart contracts run on a decentralized network, this means that the whole network is trustless.

**Use Case for Smart Contracts:**

**High Securities:**

Another one of the useful smart contract real use cases include securities. With smart contracts, capitalization table management can be simplified and improved. This means that there are no intermediaries between the parties, including security custody chains. It can also be used for dividends, automatic payments, liability management, and stock splits.

Also, smart contracts can help reduce operational risk and make workflows digitized.

**Loans and Mortgages:**

Smart contracts can also help improve financial services, including mortgages and loans. To do so, it can connect the parties and ensure that the whole process can be completed in a friction-less way. Moreover, it also provides an error-free process. For instance, the smart contract set up to handle a mortgage can manage it by tracking the payments and releasing the property when the whole loan is paid off.

One more benefit from using smart contracts in financial services is visibility to all the involved parties.

This use case also falls under the Ethereum smart contract use cases.

**Government:**

Smart contracts help automate. That is where it can help the government to manage operations. One of those operations includes land title recording where the government can use to do property transfers.

Land Title Recording requires parties to transfer property with efficiency and transparency. Smart contracts can help do so. Also, using it will reduce auditing costs and also improve transparency within the whole system.

Another use-case for government, including electronic elections, the digital identity that we discussed earlier, and electronic record filing.

Governments can explore more smart contract use cases based on their requirement.

**Insurance:**

Insurance has always been one of the most use-cases of smart contracts. It is a known fact that most of the disputes happen in the insurance sector. For example, let us take auto insurance as an example. Here, smart contracts can be used to settle the insurance as soon as possible.

To do so, smart contracts need to utilize a lot of technology, including Internet-of-Things, to facilitate themselves. The smart contract will facilitate the policy and make sure that it has all the proper documentation, including driver reports and driving records, with the use of the technology. If the smart contract is set up with the right policy, documents, and ways to capture data, it can execute itself shortly after the accident. Also, smart contract execution is only done based on the collected data, which ensures that no fraud is done in the process.

The insurance sector can genuinely take advantage of smart contracts and hence is one of the best smart contracts use cases out there. More so, it’s a smart contract application example.