VEHICLE INFORMATION SYSTEM USING BLOCKCHAIN

Amey Zulkanthiwar
California State University - San Bernardino, zulkanthiwar.amey@gmail.com

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd
Part of the Education Commons, and the Engineering Commons

Recommended Citation
Zulkanthiwar, Amey, "VEHICLE INFORMATION SYSTEM USING BLOCKCHAIN" (2019). Electronic Theses, Projects, and Dissertations. 899.
https://scholarworks.lib.csusb.edu/etd/899

This Project is brought to you for free and open access by the Office of Graduate Studies at CSUSB ScholarWorks. It has been accepted for inclusion in Electronic Theses, Projects, and Dissertations by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
VEHICLE INFORMATION SYSTEM USING BLOCKCHAIN

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Amey Pramod Zulkanthiwar
June 2019
VEHICLE INFORMATION SYSTEM USING BLOCKCHAIN

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by
Amey Pramod Zulkanthiwar

June 2019

Approved by:
Dr. Owen J. Murphy, Advisor, Computer Science and Engineering
Dr. Ernesto Gomez, Committee Member
Dr. Tong Lai Yu, Committee Member
ABSTRACT

The main purpose of a vehicle information system using blockchain is to create a transparent and reliable information system which will help consumers buy a vehicle; it is a vehicle information system.

The blockchain system will create a time sequence chain of events database for each vehicle from the original sale. It will include insurance, vehicle repair, and vehicle resale.

This project is mainly divided into three parts. Part one is used by the administration who will create the blockchain and will give authentication to a different organization to create the blockchain. Part two will be used by the Organization to create a block in the blockchain. Part three will be used by customers who want to get information about the vehicle.
ACKNOWLEDGEMENTS

I would like to thank my project adviser Dr. Owen J. Murphy and the committee members Dr. Tong Lai Yu and Dr. Ernesto Gomez for believing in me. I am greatly thankful for the confidence they have put in me while working on this project, which involves new technology and also giving me the chance to enrich and expand my knowledge of blockchain and a Smart Contract.

I am also thankful to my colleague from Patient Care Analytics company where I was given the opportunity to do my internship which helped me learn the new front-end technologies and the basics of web application development.

Last but not the least, I would like to thank my parents and family members who have supported me both financially and emotionally. None the less, I am the first person in my family to acquire a master’s degree from an international university.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>CHAPTER ONE: INTRODUCTION</td>
<td>v</td>
</tr>
<tr>
<td>Background</td>
<td>1</td>
</tr>
<tr>
<td>Purpose</td>
<td>1</td>
</tr>
<tr>
<td>CHAPTER TWO: SYSTEM ANALYSIS</td>
<td></td>
</tr>
<tr>
<td>Proposed System</td>
<td>2</td>
</tr>
<tr>
<td>System Requirement Specification</td>
<td>4</td>
</tr>
<tr>
<td>System Requirement</td>
<td>4</td>
</tr>
<tr>
<td>Hardware Requirements</td>
<td>4</td>
</tr>
<tr>
<td>Software Requirements</td>
<td>4</td>
</tr>
<tr>
<td>User Requirement</td>
<td>5</td>
</tr>
<tr>
<td>Hardware Requirements</td>
<td>5</td>
</tr>
<tr>
<td>Software Requirements</td>
<td>5</td>
</tr>
<tr>
<td>Software Used</td>
<td>5</td>
</tr>
<tr>
<td>Metamask</td>
<td>5</td>
</tr>
<tr>
<td>Firebase</td>
<td>7</td>
</tr>
<tr>
<td>Infura</td>
<td>8</td>
</tr>
<tr>
<td>Node.js</td>
<td>9</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1. Block of blockchain ................................................................. 2
Figure 2. Simplified blockchain ............................................................. 3
Figure 3. Metamask structural diagram .................................................. 6
Figure 4. Infura Integration ................................................................. 8
Figure 5. Javascript Document Object Module (DOM) ............................ 10
Figure 6. The use case diagram of Admin interaction .............................. 12
Figure 7. Interaction of Authenticated people to web application ............ 12
Figure 8. A different organization which will get authentication ............. 13
Figure 9. User interaction with web application .................................... 14
Figure 10. All interaction to Smart Contract ........................................ 14
Figure 11. All technology interaction ................................................... 15
Figure 12. Solidity version .................................................................. 16
Figure 13. Contract creation in solidity .............................................. 16
Figure 14. Access control code ............................................................ 17
Figure 15. Struct Showroom datatype .................................................. 17
Figure 16. Basic datatypes in solidity .................................................. 19
Figure 17. Datatype for storing Insurance data ...................................... 19
Figure 18. Datatype for storing DMV data ............................................ 20
Figure 19. Datatype for storing Mechanic data ..................................... 22
Figure 20. Authentication function in Smart Contract ............................. 23
Figure 21. Calling mapping from a function ........................................ 25
Figure 2. Function calling mapping for removing authentication of Ethereum address

Figure 3. Function which is going to use by authorised organization

Figure 4. Function which will call by consumers

Figure 5. Firebase set up file

Figure 6. createUserWithEmailAndPassword method

Figure 7. sendPasswordResetEmail method

Figure 8. signOut method

Figure 9. signinWithEmailAndPassword method

Figure 10. Once method from the firebase database service

Figure 11. Set method from the firebase database service

Figure 12. Administration login page

Figure 13. Organization login page

Figure 14. Showroom section of administration project

Figure 15. Insurance section of administration project

Figure 16. DMV section of administration project

Figure 17. Mechanic section of administration project

Figure 18. All user detail section of administration project

Figure 19. Insurance page from organization project

Figure 20. Showroom page from organization project

Figure 21. DMV page from organization project

Figure 22. Mechanic page from organization project

Figure 23. Consumer project front page
CHAPTER ONE

INTRODUCTION

Background

• In the United States, around 95 percent of people own a vehicle and 65 percent of them own used vehicles.

• When people want to purchase a vehicle they use websites like Autotrader, Craigslist and dealer websites.

• Buyers can obtain limited records on vehicles using services like Carfax.

Purpose

• When people buy a used vehicle, people always want to get the best deals.

• People want a vehicle that is in good conditions and that is priced well.

• When the buyer is purchasing a vehicle normally he wants all related information about the vehicle including how the vehicle was used and how many times per year the vehicle was serviced to give the buyer a little insight of the amount of money the vehicle may require in the future.

• My idea is to build a transparent and reliable system using blockchain technology where people can buy their vehicle with certainty and not be a victim of fraud.

• This system will help reduce corruption and illegal activities.
CHAPTER TWO
SYSTEM ANALYSIS

Proposed System

This web application will help people to obtain all the information related to vehicles. The main part of the application is developed using a Smart Contract which is one of the applications of Blockchain. It is an append type database which use cryptography. In a blockchain data is stored in terms of block and block consists of data, hash of the data and hash of previous block.

Data
Hash
Hash of previous block

Figure 1. Block of blockchain

The data in the blockchain is a double linked list. After storing the data, if someone tries try to change it, the link gets broken. Blockchain data is available globally, everybody contains a copy of it.
Figure 2. Simplified blockchain

In the blockchain, the hash of one block is one of the inputs to the second block and hash of the second block is one of the inputs for the third block and this goes on and because all the data is linked to one another.

**Ethereum**

- Ethereum is a blockchain service provider.
- In this project we are using the Ethereum blockchain.
- Using Smart Contract we need to interact with the Ethereum blockchain.

**Smart Contract and Solidity**

- A Smart Contract is a protocol.
- It consists of contracts, functions, and methods.
- Smart Contract creates a block in the blockchain.
- Smart Contract is written in the solidity programing language.
- In this web application, we are creating a Smart Contract which mainly has three functions.
• Function one is used by the administration who will create the blockchain and will give authentication to a different organizations to create the blockchain.

• Function two will be used by Organizations to create a block in the blockchain.

• Function three will be used by the customer who wants to get information about the vehicle.

**Web3**

• Web3 is a javascript library.

• Web3 bridge between solidity and the javascript which means it helps us to interact with the Smart Contract from the front end of the web application.

---

**System Requirement Specification**

**System Requirement**

**Hardware Requirements**

• Laptop or PC for development and testing the web application.

**Software Requirements**

• Operating system: Windows 10, Mac OS X

• Code editor: Visual studio code and Remix

• Browser: Google chrome

• Chrome Extension: Metamask

• Database: Smart contract and firebase
User Requirement

Hardware Requirements

• Laptop or PC Company: Any
• Ram: 2 GB

Software Requirements

• Operating system: Windows 10, Mac OS X
• Browser: Google chrome

Software Used

Metamask

• Metamask is a wallet for keeping Ether, which is a cryptocurrency developed by Ethereum.
• Metamask interacts with the Ethereum blockchain and is a service which is used with regular browsers, such as Chrome, Firefox, Internet Explorer, etc.
• There are some browsers like Mist and Dapps which are compatible with Ethereum blockchain.
• Ether transaction on Ethereum’s main network defines the compatibility with Ethereum blockchain.
• We cannot interact with the Ethereum blockchain with our regular browser because the regular browser does not contain any Ether.
• Metamask injects the web3 javascript library into the browser. [9]
• When someone wants to interact with the Ethereum blockchain using a regular browser.

• Metamask holds the Ether and the browser holds the data and combined they interact with the Ethereum blockchain.

• Metamask contains one Main network and three test networks (Ropsten, Kovan, and Rinkbey).

• Rinkbey will be used for testing our web application.

Figure 3. Metamask structural diagram
Firebase

Firebase Real-time Database, an API that synchronizes application information crosswise over iOS, Android, and Web gadgets, and stores it on the Firebase cloud. Firebase helps developers build a collaborative web application which is also a real-time application.

Using Firebase, developers can develop a web application which involves authentication, real-time database, storage, hosting, ML kit (Mobile machine learning system) and cloud messaging.

- Firebase authentication is a service which we can authenticate the user by their social login provider like Gmail, Facebook, Twitter, etc.
- In Firebase, real-time database data is stored in the Firebase cloud and it provides API (Application program interface) which deals with asynchronous data.
- Firebase has client libraries which allow interaction with the Node.js, Javascript, Java, iOS, and Android.
Infura

- To do the transaction in the Ethereum network we need to be a part of a network.
- Infura has a running node which is a part of the Ethereum network.
- We need to connect to the running node with our web application so that we can interact with the Ethereum network.
- Infura provides nodes for 4 networks, one is the main network node and three test network nodes Rinkbey, Kovan, and Ropsten.\(^{14}\)
- Infura provides secure and easy access to the Ethereum network.
- When we sign up on the infura website it gives us a token, and it is used to connect to the corresponding Ethereum network.

![Infura Integration Diagram](image)

Figure 4: Infura Integration

- In the project, we are using a Rinkbey and a main infura service provider to connect to the Rinkbey test network and the Main network of Ethereum.
Node.js

- Node Js is an extension of the javascript language and it is a runtime open source server environment.
- It is used to run javascript on the server.

Technology Used

Ethereum

- Ethereum uses the decentralized and distributed network.
- Ethereum network store a data called blockchain in each system/node of a network.
- To store data into the Ethereum blockchain we need a Smart Contract which is written in the solidity language.
- When we execute a Smart Contract, at that time it stores data into the blockchain.

Programming Languages Used

JavaScript

- Javascript is a client-side script.
- In this project, we used javascript for web development to handle client-side validation and client-side interaction.
- Javascript uses a document object module (DOM) manipulation.
• Javascript provides the properties and methods that are used to manipulate the document elements such as images, links, forms, etc.

• Javascript document object module diagram is below.

![Diagram of Javascript Document Object Module (DOM)](image)

Figure 5: Javascript Document Object Module (DOM)
Solidity

• Solidity is an object-oriented programming language which is used for writing a Smart Contract for Ethereum.

• Solidity is developed by the core team member of Ethereum.

• Solidity was influenced by JavaScript, C++, and Python and is intended to focus on the Ethereum Virtual Machine (EVM).

• A Remix is a code editor where we can write a Smart Contract in solidity and it has all the version of the compiler is available.

• Solc is a compiler that we used for compiling solidity code if we are not using the remix code editor.

• The latest version of solidity available is v0.5.7.[17]
CHAPTER THREE

UML DIAGRAM

Use Case Diagram

Use case diagram display the relationships between the users and the applications.

Figure 6. Use case diagram of Admin interaction

The above diagram represents the relation of Admin with the Smart Contract and web application. Admin needs to deploy Smart Contract separately and then interact with the Smart Contract through the web application.

Figure 7. Interaction of Authenticated people to the web application
The above diagram represents the relation of authenticated users with the Smart Contract through the web application. Admin is responsible for designating users so that authenticated users can add the information into the Smart Contract. Admin gives authorization to the different organizations. The use case diagram is shown below.

![Diagram](image_url)

**Figure 8. Different organization which will get authentication**

Data is added by an authenticated organization, and after that users will use the web application to get the data. Its use case diagram is shown below.
First, the Admin will deploy the Smart Contract on the Ethereum network and then Organizations and Users will interact with it through the web application. Only one Smart Contract is deployed, but it has a different functionality for different purposes.
All Technology Interaction Diagram

The interaction between all the technology that is used in the web application is given below.

Figure 11. All technology interaction
SMART CONTRACT

• Smart Contract is a protocol used to store the data into the Ethereum blockchain.

**Focusing on the coding part for the Smart Contract.**

• In the first line of code, we mentioned what version we are using

```
pragma solidity 0.4.24;
```

**Figure 12. Solidity version**

• The next step is to create a contract, Creating a contract is like creating a class or function. In solidity if we want to create a contract, we need to use "contract" keyword.

```
import "./AccessControl.sol";

contract VehicleVerification is Ownable {
```

**Figure 13. Contract creation in solidity**

• Access Control code is given below.
pragma solidity ^0.4.24;

/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */

class Ownable {
    address private _owner;

    /**
     * @dev The Ownable constructor sets the original `owner` of the contract to the sender
     * account.
     */
    constructor() public {
        _owner = msg.sender;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner());
        ::
    }

    /**
     * @dev true if `msg.sender` is the owner of the contract.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }
}

struct showroomData {
    bytes32 vin;
    bytes32 ownerDetail;
    bytes32 modelDetail;
    bytes32 carCost;
    bytes32 carLoanOrBankDetail;
    bytes32 showroomName;
    bytes32 showroomAddress;
    bytes32 showroomId;
    uint256 issuedDate;
    bytes32 additionalDetail;
}
ShowroomData is a struct datatype and it consists of the variables which are described below.

- vin is used for storing the vin number of the vehicle.
- ownerDetail is used for storing the details of the owner’s vehicle.
- modelDetail is used for storing the vehicle details.
- carCost is used to store the price of a vehicle.
- carLoanOrBankDetail is used for storing the loan details for that particular vehicle.
- showroomName is used for storing the authorized showroom name.
- showroomAddress is used for storing the authorized showroom address.
- showroomId is used for storing the id which is generated by the firebase authentication.
- issuedDate is used for storing the time in which data is stored.
- additionalDetails is used for storing the additional details that showroom people want to add.

Basic data types used in Solidity are given below.
In Smart Contract there are four struct data types created: ShowroomData, insuranceData, motorDeptData, and mechanicData.

insuranceData is a struct datatype and it consists of the variables which are described below.

- `vin` is used to store the vin number of the vehicle.
- `insuranceNumber` is used to store the insurance policy number.
- **numOfInsurance** is used to store the number of people who is going to have insurance on the vehicle.
- **insuranceDetail** is used to store the type of insurance.
- **insurancId** is used to store the insurance id which is generated by Firebase.
- **insuranceCompanyName** is used to store the authorized insurance company name.
- **insuranceCompanyAddress** is used to store the authorized insurance company address.
- **issuedDate** is used to store the date the policy is issued.
- **additionalDetail** is used to store additional details that insurance company people want to add.

```c
struct motorDeptData {
    bytes32 vin;
    bytes32 ownerName;
    bytes32 regId;
    bytes32 milesOnCar;
    bytes32 modelDetail;
    bytes32 numberPlate;
    bytes32 motorDeptCity;
    bytes32 motorDeptAddress;
    uint256 issuedDate;
    bytes32 additionalDetail;
}
```

Figure 18. Datatype for storing DMV data

- `motorDeptData` is a struct datatype and it consists of the variables which are described below.
o vin is used to store the vin number of vehicle.

o ownerName is used to store the owner name of vehicle.

o regId is used to store the registration Id of vehicle.

o milesOnCar is used to store the miles of vehicle.

o modelDetail is used to store the model details of the vehicle like name, year, number of doors.

o numberPlate is used to store the number plate of the vehicle.

o motorDeptCity is used to store the city name of particular DMV which is adding this data.

o motorDeptAddress is used to store the address of the DMV which is storing this data into the blockchain.

o issuedDate is used to store the time in which the data is going to stored.

o additionalDetail is used to store any additional details that dmv people want to add.

• mechanicData is struct datatype and it consists of the variables which are described below.

  o vin is used to store the vin number of vehicle.

  o ownerName is used to store the name of vehicle owner.

  o shopId is used to store the Id of the mechanic shop.

  o shopName is used to store the name of the mechanic shop.

  o shopAddress is used to store the address of the shop.
- carDetail is used to store the model details of the vehicle, such as the name, year, and number of doors.
- workDescription is used to store the work that was done on the vehicle by the mechanic.
- moneyCharged is used to store the money which is charged by the mechanic for repairing the vehicle.
- typeOfWork is used to store the type of work that was done on the vehicle like oil change, tire rotation, etc.
- IssuedDate is used to store the date.
- additionalDetail is used for storing additional details that insurance company people want to add.

```
struct mechanicData {
    bytes32 vin;
    bytes32 shopId;
    bytes32 shopName;
    bytes32 shopAddress;
    bytes32 carDetail;
    bytes32 workDescription;
    bytes32 moneyCharged;
    bytes32 typeOfWork;
    uint256 issuedDate;
    bytes32 additionalDetail;
}
```

Figure 19. Datatype for storing mechanic data

As we discussed earlier, there are three types of functions, First we will start with the function which is used by the admin, also known as the contract owner.
In the above figure, there are two types of functions:

- Function type one gives authentication to a particular Ethereum address for adding data.
  - Subtype: addShowroomOwner, addInsuranceOwner, addMotorDeptOwner and addMechanicOwner
• Function type two is used to remove the authentication from the particular Ethereum address.
  
  o Subtype: removeShowroomOwner, removeInsuranceOwner, removeMotorDeptOwner and removeMechanicOwner

This function can only be called the Smart Contract creator or admin. There is one function modifier called “onlyOwner” added in front of all the functions. A function modifier always runs before the function in which we use. It checks whether the function is called by Smart Contract creator or not, if it is not called by the Smart Contract creator then this function will not be executed. However, if the function is called by the Smart Contract creator then only it allows this function to run.

• An address is passed as a parameter to this function.

• Now, let's focus on what is inside the function. Inside the function, another mapping function is written, which is very efficient for looking up data in Solidity language. The mapping function stores true value for that particular address.
For removing the authentication of Ethereum address, the Boolean value of that Ethereum address needs to set as a false, as shown in the figure below.

Figure 21. Calling mapping from a function

Now move towards the type two functions which are going to be used by the authorized organization for adding data into the Ethereum blockchain.

- All the data which is going to be the part of the blockchain is needed to pass as an argument.
- The first line of code in this function is the required function, which will make sure only authorized organization can call this function.
- The second line of code is used to create a unique hash of the provided data.

```
// Adding records by mechanic
function addMechanicData(bytes32 vin, bytes32 shopId, bytes32 shopName, bytes32 shopAddress, bytes32 carDetail, bytes32 vinDescription, bytes32 moneyCharged, bytes32 typeOfWork, uint256 issuedDate, bytes32_additionalDetail) public returns (bytes32) {
    // Authentication: whether the address that is adding data has authority or not
    require(MechanicOwners[msg.sender] == true);

    // Taking a unique hash of data
    bytes32 mechanicDataId = keccak256(dbc.encodePacked(vin, shopId, shopName, shopAddress, carDetail, vinDescription, moneyCharged, typeOfWork, issuedDate, additionalDetail));

    // Saving id to mechanicData
    mechanicData[mechanicDataId] = mechanicData(vin, shopId, shopName, shopAddress, carDetail, vinDescription, moneyCharged, typeOfWork, issuedDate, additionalDetail);

    // Mapping vin number to mechanic Data id
    myMechanicData[vin].push(mechanicDataId);

    // Pushing mechanicDataId to array
    mechanicDataIds.push(mechanicDataId);

    // Mapping unique id of every record to Data
    // Here we are mapping a unique id of each record.
    mapping(bytes32 => showRoomData) public idToShowRoomData;
    mapping(bytes32 => insuranceData) public idToInsuranceData;
    mapping(bytes32 => motorDeptData) public idToMotorDeptData;
    mapping(bytes32 => mechanicData) public idToMechanicData;

    // Mapping vin to unique id of record
    // Since we can't remember unique id of data.
    // We are mapping vin number to the unique id of data
    mapping(bytes32 => bytes32[]) public vinToShowRoomDataId;
    mapping(bytes32 => bytes32[]) public vinToInsuranceDataId;
    mapping(bytes32 => bytes32[]) public vinToMotorDeptDataId;
    mapping(bytes32 => bytes32[]) public vinToMechanicDataId;

    // Saving unique ids of data into the array
    bytes32[] public showRoomDataIds;
    bytes32[] public insuranceDataIds;
    bytes32[] public motorDeptDataIds;
    bytes32[] public mechanicDataIds;
```

Figure 23. Function which is going to use by authorised organization

- Assign/Point all the values that we are passed to that function to the hash code that we just generated using keccak256, with the help of mapping as shown in the figure above (Follow the arrows to understand).
• Now, assign hash value to the vin number of the vehicle.
• Any particular vehicle will always have the same vin number, but every time changes are made to some parts of the data.
• Vin number will act as an index for the vinToMechanicDataId array of mapping.
• vinToMechanicDataId is an array of hash values and index for vinToMechanicDataId is a vin number.
• The last part is to return from the Smart Contract.

Now we will consider the function used to get information from the Ethereum blockchain.

In this particular type, we have written eight functions for four organizations and each organization type has two functions, but all the functions work similarly.

There is a limit to the number of variables that we can return in the Solidity programing language, and that is a reason behind writing two functions for each organization, so here we divided the return variables in half.
Now we will discuss the `getMechanicDataByVin1` function which is used for obtaining information from the Ethereum blockchain.

- We are going to pass the Vin Number as an argument to this function and will get the information in the form of arrays.
- First, we will collect all the data id which is linked to the given vin number, we will get data in the form of the array because the way its stored.
- It is an array of data id’s (identity documents), each id points towards a unique data because of the mapping.
- Now, we are using the for loop to go through each id.
- Inside the array, we are getting data with the help of id and storing that data into each array which we will later return.
- Comments are given in the code which will help while referencing.

```java
// Adding data by virtual insurance people
function addInsuranceData(bytes32 vin, bytes32 _insuranceNumber, bytes32 _insuranceName, bytes32 _insuranceDetail, bytes32 _insuranceId, bytes32 _insuranceCompanyName, bytes32 _insuranceCompanyAddress, uint64 _issueDate, bytes32 _additionalDetail) public returns (bytes32) {

// this function is used by the client app to get the records
function getMechanicDataByVin1(bytes32 _vin) public view returns (bytes32[], bytes32[], bytes32[], bytes32[], bytes32[])
{

// Setting mechanicDataId by searching through mapping
bytes32[] mechanicDataID = vinToMechanicDataId[_vin];

// Initializing data array to return containing data
bytes32[] memory vin = new bytes32[] (mechanicDataId.length);
bytes32[] memory shopID = new bytes32[] (mechanicDataId.length);
bytes32[] memory shopName = new bytes32[] (mechanicDataId.length);
bytes32[] memory shopAddress = new bytes32[] (mechanicDataId.length);
bytes32[] memory carDetail = new bytes32[] (mechanicDataId.length);

// looping through mechanicDataId and saving into the above data array
for (uint i = 0; i < mechanicDataId.length; i++) {

// datatype: storagetype: varname; mechanicData memory data = idToMechanicData[mechanicDataId[i]];
vin[i] = data.vin;
shopID[i] = data.shopID;
shopName[i] = data.shopName;
shopAddress[i] = data.shopAddress;
carDetail[i] = data.carDetail;
}
return (vin, shopID, shopName, shopAddress, carDetail);
}
```
Firebase

In this section of the document, we will talk about the authentication and the Realtime database that we used from the firebase service for our application. The general explanation for the firebase is explained in chapter two, software used section of this document. In the part of the document, we have explained how firebase is used in the application.

- First, we need to set up the firebase in our application, and to setup the firebase we need to create a new project in the firebase website.
- After creating a new project in firebase, it will give an option of a console which will contain all configuration details. You can see our project credential in the figure below.

```javascript
import firebase from "firebase/app";
import "firebase/database";
import "firebase/auth";

//credential which we will get from the firebase
var config = {
  apiKey: "AIzaSyB_q1ZKjxF1ce0Qz2UFUDEfenbKAASL0g",
  authDomain: "vehical-information-syatem.firebaseapp.com",
  databaseURL: "https://vehical-information-syatem.firebaseio.com",
  projectId: "vehical-information-syatem",
  storageBucket: "vehical-information-syatem.appspot.com",
  messagingSenderId: "82056055207"
};

firebase.initializeApp(config);

//this is for the real time database in firebase
export const db = firebase.database();

// this is for the authentication purpose in firebase
export const auth = firebase.auth();
```

Figure 25. Firebase set up file
• As we can see in the figure shown above, we have imported the service interface of firebase authentication and database module and later in the code we have exported the service so that we can use the method within these services, whenever needed.

• For our project, we have used the email authentication. We can also use multiple authentications if we like.

In this project we are using four methods from the Firebase authentication service interface createUserWithEmailAndPassword, sendPasswordResetEmail, signOut, and signInWithEmailAndPassword. createUserWithEmailAndPassword is the one used in the administration project. The admin person will use this method to create login details for different organizations. The following figure will show how we exactly use this method.

```
auth.createUserWithEmailAndPassword(this.state.email, this.state.password)
.then(res => {
  this.submitTransaction();
  let uid = auth.currentUser.uid;
  console.log('work');
  let obj = this.state;
  obj.address = obj.street + ',' + obj.city + ',' + obj.states + ',' + obj.country + ',' + obj.zip;
  delete obj.street;
  delete obj.city;
  delete obj.states;
  delete obj.country;
  delete obj.zip;
  console.log('obj.address', obj.address);
  obj.type = 'insurance'
  db.ref('/users/' + uid).set(obj)
    .then(res => { })
    .catch(err => alert(err.message))
})
.catch(err => {
  alert(err.message)
})
```

Figure 26. createUserWithEmailAndPassword method
For this method we need to pass the email and password information as an argument. If this method runs perfectly then the code with in “then” will execute and if it fails then the code with in “catch” will execute.

The sendPasswordResetEmail method will be used by different organization to reset the password if any users forget their passwords. Image of this function is below.

```
auth.sendPasswordResetEmail(this.props.email)
  .then(function() {
    // Email sent.
    alert("Password reset email has been sent");
  })
  .catch(function(error) {
    // An error happened.
  });
```

Figure 27. The sendPasswordResetEmail method

The signOut method will be used for signing out of the web application. This method will be used by the administrator and the organization. The image of this function is given below.

```
auth.signOut()
  .then(function() {
    that.props.logout()
    console.log("successful")
    history.push('/')
    // Sign-out successful.
  })
  .catch(function(error) {
    // An error happened
  });
```

Figure 28. The signOut method
signinWithEmailAndPassword method will be used for login purposes, this method will be used by the administrator and the organization. Image of the function is given below.

```javascript
auth.signInWithEmailAndPassword(this.state.email, this.state.password)
  .then(res => {
    let user_uid = res.user.uid;
    db.ref(`/admin`).once("value", snapshot => {
      if (user_uid == snapshot.val().uid) {
        this.props.signInIn();
      } else {
        this.setState({
          errorTextPass: "this user doesn’t have admin rights."
        });
      }
    });
  })
  .catch(err => {
    // not able to sign in then this part will work
    console.log("error", err);
    //checking because of what we couldn't able to login
    if (err.message.includes("email")) {
      this.setState({
        errorTextEmail: err.message
      });
    } else if (err.message.includes("password")) {
      this.setState({
        errorTextPass: err.message
      });
    } else {
      this.setState({
        error: err.message
      });
    }
  });
```

Figure 29. The signInWithEmailAndPassword method

Now for the Firebase database service interface, we are using database service and methods within it to store the user information which is provided by the administration. For storing purpose, we have used set method and for retrieving purpose we have used the once method. Images of this method are shown below.
let user_uid = firebase.uid

```
db.ref('/admin').once("value", snapshot => {
    if (user_uid == snapshot.val().uid)
    {
        this.props.signIn();
    }
    else {
        this.setState({
            errorTextPass: "this user doesn't have admin rights."
        })
    }
})
```

Figure 30. Once method from the firebase database service

```
obj.type = 'department'

db.ref(`/users/` + uid).set(obj)
    .then((res) => {
    })
    .catch(err => alert(err.message))
```

Figure 31. Set method from the firebase database service
CHAPTER FIVE

SCREENSHOTS OF PROJECT

In this section of the document, screenshots of this project are shown. The images present below are of the login system for the administration and organization.

Figure 32. Administration login page
Figure 33. Organization login page

Now below we can see all the screenshots related to the administration project.

Figure 34. Showroom section of administration project
Figure 35. Insurance section of the administration project

Figure 36. DMV section of the administration project
Figure 37. Mechanic section of the administration project

Figure 38. All user detail section of the administration project

Now below we can see all the screenshots of the organization project.
Figure 39. Insurance page from the organization project

Figure 40. Showroom page from the organization project
Now last but not the least is screenshots of consumer project is shown below.
Figure 43. Consumer project front page
CHAPTER SIX
FUTURE ENHANCEMENTS

Deleting Authentication From Administration Project

With the current web application, the administration can take out the authorization of any organization from a Smart Contract and delete their database from the Firebase, but would not be able to delete their login authentication details from the Firebase. Authentication details can either be deleted from the Firebase website or users need to login into the system to delete the account. The administrator cannot delete the authentication details of the user from the Administration web application. The Firebase did not provide any methods for deleting authentication from the administration side.
CHAPTER SEVEN

CONCLUSION

In this Vehicle Verification Using Blockchain Concept project, we have successfully developed a web application which will add the data into the blockchain; however, the data, once added to the blockchain, will not be able to update or modify, which was the primary goal of the project. We were able to build a web application with the login system which will interact with the blockchain. In the vehicle verification project, we were able to combine the centralized system with the decentralized and distributed system. We were able to achieve double authentication for the web application using Firebase and Metamask, which increases the security of web application and integrity of the data which is going to be added into the blockchain.
APPENDIX

IMPORTANT CODE OF PROJECT
pragma solidity 0.4.24;

import "./AccessControl.sol";

contract VehicleVerification is Ownable {

    struct showroomData {
        bytes32 vin;
        bytes32 ownerDetail;
        bytes32 modelDetail;
        bytes32 carCost;
        bytes32 carLoanOrBankDetail;
        bytes32 showroomName;
        bytes32 showroomAddress;
        bytes32 showroomId;
        uint256 issuedDate;
        bytes32 additionalDetail;
    }

    struct insuranceData {
        bytes32 vin;
        bytes32 insuranceNumber;
        bytes32 numOfInsurance;
        bytes32 insuranceDetail;
        bytes32 insuranceId;
        bytes32 insuranceCompanyName;
        bytes32 insuranceCompanyAddress;
        uint256 issuedDate;
        bytes32 additionalDetail;
    }

    struct motorDeptData {
        bytes32 vin;
        bytes32 ownerName;
        bytes32 regId;
        bytes32 milesOnCar;
        bytes32 modelDetail;
        bytes32 numberPlate;
        bytes32 motorDeptCity;
        bytes32 motorDeptAddress;
        uint256 issuedDate;
        bytes32 additionalDetail;
    }
struct mechanicData {
  bytes32 vin;
  bytes32 shopId;
  bytes32 shopName;
  bytes32 shopAddress;
  bytes32 carDetail;
  bytes32 workDescription;
  bytes32 moneyCharged;
  bytes32 typeOfWork;
  uint256 issuedDate;
  bytes32 additionalDetail;
}

// Saving addresses of data creators
mapping(address => bool) public ShowroomOwners;
mapping(address => bool) public InsuranceOwners;
mapping(address => bool) public MotorDeptOwners;
mapping(address => bool) public MechanicOwners;

// Mapping unique id of every record to Data
mapping(bytes32 => showroomData) public idToShowRoomData;
mapping(bytes32 => insuranceData) public idToInsuranceData;
mapping(bytes32 => motorDeptData) public idToMotorDeptData;
mapping(bytes32 => mechanicData) public idToMechanicData;

// Mapping vin to unique id of record
mapping(bytes32 => bytes32[]) public vinToShowroomDataId;
mapping(bytes32 => bytes32[]) public vinToInsuranceDataId;
mapping(bytes32 => bytes32[]) public vinToMotorDeptDataId;
mapping(bytes32 => bytes32[]) public vinToMechanicDataId;

// Saving unique ids of data
bytes32[] public showroomDataIds;
bytes32[] public insuranceDataIds;
bytes32[] public motorDeptDataIds;
bytes32[] public mechanicDataIds;

// Adding owner functions
function addShowroomOwner(address _addr) public onlyOwner {
  ShowroomOwners[_addr] = true;
}
function addInsuranceOwner(address _addr) public onlyOwner {
    InsuranceOwners[_addr] = true;
}

function addMotorDeptOwner(address _addr) public onlyOwner {
    MotorDeptOwners[_addr] = true;
}

function addMechanicOwner(address _addr) public onlyOwner {
    MechanicOwners[_addr] = true;
}

//Removing owner functions
function removeShowroomOwner(address _addr) public onlyOwner{
    ShowroomOwners[_addr] = false;
}

function removeInsuranceOwner(address _addr)public onlyOwner{
    InsuranceOwners[_addr] = false;
}

function removeMotorDeptOwner(address _addr)public onlyOwner{
    MotorDeptOwners[_addr] = false;
}

function removeMechanicOwner(address _addr) public onlyOwner{
    MechanicOwners[_addr] = false;
}

function addMechanicData(bytes32 _vin, bytes32 _shopId , bytes32 _shopName, bytes32 _shopAddress, bytes32 _carDetail , bytes32 _workDescription, bytes32 _moneyCharged, bytes32 _typeOfWork, uint256 _issuedDate, bytes32 _additionalDetail) public  returns (bytes32) {
    require(MechanicOwners[msg.sender] == true);
    bytes32 mechanicDataId = keccak256(abi.encodePacked(_vin, _shopId, _shopName, _shopAddress, _carDetail, _workDescription, _moneyCharged, _typeOfWork, _issuedDate, _additionalDetail));
    idToMechanicData[mechanicDataId] = mechanicData(_vin, _shopId, _shopName, _shopAddress, _carDetail, _workDescription, _moneyCharged, _typeOfWork, _issuedDate, _additionalDetail);
}
```solidity
def addMotorDeptData(bytes32 _vin, bytes32 _ownerName, bytes32 _regId, bytes32 _milesOnCar, bytes32 _modelDetail, bytes32 _numberPlate, bytes32 _motorDeptCity, bytes32 _motorDeptAddress, uint256 _issuedDate, bytes32 _additionalDetail) public returns (bytes32) {
    require(MotorDeptOwners[msg.sender] == true);
    bytes32 motorDeptDataId = keccak256(abi.encodePacked(_vin, _ownerName, _regId, _milesOnCar, _modelDetail, _numberPlate, _motorDeptCity, _motorDeptAddress, _issuedDate, _additionalDetail));
    idToMotorDeptData[motor_dept_dataId] = motorDeptData(_vin, _ownerName, _regId, _milesOnCar, _modelDetail, _numberPlate, _motorDeptCity, _motorDeptAddress, _issuedDate, _additionalDetail);
    vinToMotorDeptDataId[_vin].push(motor_dept_dataId);
    motorDeptDataIds.push(motor_dept_dataId);
    return motor_dept_dataId;
}
```

```solidity
def addShowroomData( bytes32 _vin, bytes32 _ownerDetail, bytes32 _modelDetail, bytes32 _carCost, bytes32 _carLoanOrBankDetail, bytes32 _showroomName, bytes32 _showroomAddress, bytes32 _showroomId, uint256 _issuedDate, bytes32 _additionalDetail) public returns (bytes32) {
    require(ShowroomOwners[msg.sender] == true);
    bytes32 showroomDataId = keccak256(abi.encodePacked(_vin, _ownerDetail, _modelDetail, _carCost, _carLoanOrBankDetail, _showroomName, _showroomAddress, _showroomId, _issuedDate, _additionalDetail));
    showroomToShowroomId[_vin].push(showroomId);
    showroomDataIds.push(showroomId);
    return showroomId;
}
```
idToShowRoomData[showroomDataId] = showroomData(_vin,
_ownerDetail, _modelDetail, _carCost, _carLoanOrBankDetail,
_showroomName, _showroomAddress , _showroomId,
_issuedDate, _additionalDetail);

vinToShowroomDataId[_vin].push(showroomDataId);

showroomDataIds.push(showroomDataId);

return showroomDataId;
}

function addInsuranceData(bytes32 _vin, bytes32 _insuranceNumber,
bytes32 _numOfInsurance, bytes32 _insuranceDetail , bytes32
InsuranceId , bytes32 _insuranceCompanyName, bytes32
_insuranceCompanyAddress, uint256 _issuedDate, bytes32
_additionalDetail) public  returns (bytes32) {

require(InsuranceOwners[msg.sender] == true);

bytes32 insuranceDataId = keccak256(abi.encodePacked( _vin,
_insuranceNumber, _numOfInsurance , _insuranceDetail,
_insuranceId , _insuranceCompanyName,
_insuranceCompanyAddress , _issuedDate, _additionalDetail));

idToInsuranceData[insuranceDataId] = insuranceData(_vin,
_insuranceNumber, _numOfInsurance , _insuranceDetail, _insuranceld,
_insuranceCompanyName, _insuranceCompanyAddress , _issuedDate, _additionalDetail);

vinToInsuranceDataId[_vin].push(insuranceDataId);

insuranceDataIds.push(insuranceDataId);

return insuranceDataId;
}

function getMechanicDataByVin1(bytes32 _vin) public view returns
(bytes32[], bytes32[], bytes32[], bytes32[], bytes32[]) {

bytes32[] mechanicDataId = vinToMechanicDataId[_vin];
bytes32[] memory vin = new bytes32[](mechanicDataId.length);
bytes32[] memory shopId = new bytes32[](mechanicDataId.length);
bytes32[] memory shopName = new bytes32[](mechanicDataId.length);
bytes32[] memory shopAddress = new bytes32[](mechanicDataId.length);
bytes32[] memory carDetail = new bytes32[](mechanicDataId.length);

for (uint i = 0; i < mechanicDataId.length; i++) {
    mechanicData memory data = idToMechanicData[mechanicDataId[i]];
    cin[i] = data.vin;
    shopId[i] = data.shopId;
    shopName[i] = data.shopName;
    shopAddress[i] = data.shopAddress;
    carDetail[i] = data.carDetail;
}
return (vin, shopId, shopName, shopAddress, carDetail);

function getMechanicDataByVin2(bytes32 _vin) public view returns (bytes32[], bytes32[], bytes32[], uint256[], bytes32[]) {
    bytes32[] mechanicDataId = vinToMechanicDataId[_vin];

    bytes32[] memory workDescription = new bytes32[](mechanicDataId.length);
    bytes32[] memory moneyCharged = new bytes32[](mechanicDataId.length);
    bytes32[] memory typeOfWork = new bytes32[](mechanicDataId.length);
    uint256[] memory issuedDate = new uint256[](mechanicDataId.length);
    bytes32[] memory additionalDetail = new bytes32[](mechanicDataId.length);

    for (uint i = 0; i < mechanicDataId.length; i++) {
        mechanicData memory data = idToMechanicData[mechanicDataId[i]];
        workDescription[i] = data.workDescription;
        moneyCharged[i] = data.mone

mechanicData memory data =
    idToMechanicData[mechanicDataId[i]];

workDescription[i] = data.workDescription;
moneyCharged[i] = data.moneyCharged;
typeOfWork[i] = data.typeOfWork;
issuedDate[i] = data.issuedDate;
additionalDetail[i] = data.additionalDetail;
}
return (workDescription, moneyCharged, typeOfWork, issuedDate, additionalDetail);
}

function getInsuranceDataByVin1(bytes32 _vin) public view returns (bytes32[], bytes32[], bytes32[], bytes32[], bytes32[]) {

bytes32[] insuranceDataId = vinToInsuranceDataId[_vin];

bytes32[] memory vin = new bytes32[](insuranceDataId.length);

bytes32[] memory insuranceNumber = new bytes32[](insuranceDataId.length);

bytes32[] memory numOfInsurance = new bytes32[](insuranceDataId.length);

bytes32[] memory insuranceDetail = new bytes32[](insuranceDataId.length);

bytes32[] memory insuranceId = new bytes32[](insuranceDataId.length);

for (uint i = 0; i < insuranceDataId.length; i++) {
    insuranceData memory data =
        idToInsuranceData[insuranceDataId[i]];
    vin[i] = data.vin;
    insuranceNumber[i] = data.insuranceNumber;
    numOfInsurance[i] = data.numOfInsurance;
    insuranceDetail[i] = data.insuranceDetail;
    insuranceId[i] = data.insuranceId;
}
return (vin, insuranceNumber , numOfInsurance, insuranceDetail, insuranceId);
}

function getInsuranceDataByVin2(bytes32 _vin) public view returns (bytes32[], bytes32[], uint256[], bytes32[]) {

bytes32[] insuranceDataId = vinToInsuranceDataId[_vin];

bytes32[] memory insuranceCompanyName = new bytes32[](insuranceDataId.length);

bytes32[] memory insuranceCompanyAddress = new bytes32[](insuranceDataId.length);

uint256[] memory issuedDate = new uint256[](insuranceDataId.length);

bytes32[] memory additionalDetail = new bytes32[](insuranceDataId.length);

for (uint i = 0; i < insuranceDataId.length; i++) {

insuranceData memory data =
idToInsuranceData[insuranceDataId[i]];

insuranceCompanyName[i] = data.insuranceCompanyName;

insuranceCompanyAddress[i] = data.insuranceCompanyAddress;

issuedDate[i] = data.issuedDate;

additionalDetail[i] = data.additionalDetail;
}
return (insuranceCompanyName, insuranceCompanyAddress, issuedDate, additionalDetail);
}

function getMotorDeptDataByVin1(bytes32 _vin) public view returns (bytes32[], bytes32[], bytes32[], bytes32[], bytes32[]) {

bytes32[] motorDeptDataId = vinToMotorDeptDataId[_vin];

bytes32[] memory vin = new bytes32[](motorDeptDataId.length);

bytes32[] memory ownerName = new bytes32[](motorDeptDataId.length);
bytes32[] memory regId = new bytes32[](motorDeptDataId.length);

bytes32[] memory milesOnCar = new bytes32[](motorDeptDataId.length);

bytes32[] memory modelDetail = new bytes32[](motorDeptDataId.length);

for (uint i = 0; i < motorDeptDataId.length; i++) {
    motorDeptData memory data = idToMotorDeptData[motorDeptDataId[i]];

    vin[i] = data.vin;
    ownerName[i] = data.ownerName;
    regId[i] = data.regId;
    milesOnCar[i] = data.milesOnCar;
    modelDetail[i] = data.modelDetail;
}

return (vin, ownerName, regId, milesOnCar, modelDetail);

function getMotorDeptDataByVin2(bytes32 _vin) public view returns (bytes32[], bytes32[], bytes32[], uint256[], bytes32[]) {

    bytes32[] motorDeptDataId = vinToMotorDeptDataId[_vin];

    bytes32[] memory numberPlate = new bytes32[](motorDeptDataId.length);

    bytes32[] memory motorDeptCity = new bytes32[](motorDeptDataId.length);

    bytes32[] memory motorDeptAddress = new bytes32[](motorDeptDataId.length);

    uint256[] memory issuedDate = new uint256[](motorDeptDataId.length);

    bytes32[] memory additionalDetail = new bytes32[](motorDeptDataId.length);

    for (uint i = 0; i < motorDeptDataId.length; i++) {

motorDeptData memory data =
    idToMotorDeptData[motorDeptDataId[i]];

numberPlate[i] = data.numberPlate;
motorDeptCity[i] = data.motorDeptCity;
motorDeptAddress[i] = data.motorDeptAddress;
issuedDate[i] = data.issuedDate;
additionalDetail[i] = data.additionalDetail;
}
return (numberPlate, motorDeptCity, motorDeptAddress, issuedDate, additionalDetail);

function getShowroomDataByVin1(bytes32 _vin) public view returns
    (bytes32[], bytes32[], bytes32[], bytes32[], bytes32[], bytes32[], bytes32[]) {

    bytes32[] showroomDataId = vinToShowroomDataId[_vin];

    bytes32[] memory vin = new bytes32[](showroomDataId.length);
    bytes32[] memory ownerDetail = new bytes32[](showroomDataId.length);
    bytes32[] memory modelDetail = new bytes32[](showroomDataId.length);
    bytes32[] memory carCost = new bytes32[](showroomDataId.length);
    bytes32[] memory carLoanOrBankDetail = new bytes32[](showroomDataId.length);

    for (uint i = 0; i < showroomDataId.length; i++){
        showroomData memory data =
            idToShowRoomData[showroomDataId[i]];

        vin[i] = data.vin;
        ownerDetail[i] = data.ownerDetail;
        modelDetail[i] = data.modelDetail;
        carCost[i] = data.carCost;
        carLoanOrBankDetail[i] = data.carLoanOrBankDetail;
    }
}
carLoanOrBankDetail[i] = data.carLoanOrBankDetail;
}
return (vin, ownerDetail, modelDetail, carCost, carLoanOrBankDetail);
}

function getShowroomDataByVin2(bytes32 _vin) public view returns (bytes32[], bytes32[], bytes32[], uint256[], bytes32[])
{
bytes32[] showroomDataId = vinToShowroomDataId[_vin];

bytes32[] memory showroomName = new bytes32[](showroomDataId.length);

bytes32[] memory showroomAddress = new bytes32[](showroomDataId.length);

bytes32[] memory showroomId = new bytes32[](showroomDataId.length);

uint256[] memory issuedDate = new uint256[](showroomDataId.length);

bytes32[] memory additionalDetail = new bytes32[](showroomDataId.length);

for (uint i = 0; i < showroomDataId.length; i++) {

    showroomData memory data = idToShowroomData[showroomDataId[i]];

    showroomName[i] = data.showroomName;
    showroomAddress[i] = data.showroomAddress;
    showroomId[i] = data.showroomId;
    issuedDate[i] = data.issuedDate;
    additionalDetail[i] = data.additionalDetail;
}
return (showroomName, showroomAddress, showroomId, issuedDate, additionalDetail);
}
}

AccessControl.sol

pragma solidity ^0.4.24;
contract Ownable {
    address private _owner;

    /*
     * @dev The Ownable constructor sets the original `owner` of the contract
     * to the sender
     */
    constructor() public {
        _owner = msg.sender;
    }

    /*
     * @return the address of the owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /*
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner());
        _;
    }

    /*
     * @return true if `msg.sender` is the owner of the contract.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }
}
REFERENCES

   Available: https://nodejs.org/en/docs/guides

   Available: https://reactjs.org/tutorial/tutorial.html

   Available: https://material-ui.com/getting-started/usage

   Available: https://www.npmjs.com/package/mocha

   Available: https://solidity.readthedocs.io/en/v0.5.6/

   Available: https://firebase.google.com/docs/guides/

   Available: https://www.npmjs.com/package/react-grid-system

   Available: https://www.npmjs.com/package/react-sidebar

   Available: https://web3js.readthedocs.io/en/1.0/

    Available: https://redux.js.org/introduction/getting-started

Available: https://github.com/reduxjs/redux-thunk


Available: https://infura.io/docs


Available: https://metamask.github.io/metamask-docs/


Available: https://www.w3schools.com/js/