Bit-Contracts: Smart Contracts for Cars

Submitted in partial fulfilment of the requirements of the degree of

Bachelor of Computer Engineering

BY

Ashish Roy (Roll No. 46)

Priti Negi (Roll No. 32)

Alai Parekh (Roll No. 36)

Guide

Dr. Vinayak D. Shinde

Department Of Computer Engineering

Shree L. R. Tiwari College Of Engineering

Kanakia Park, Mira Road (E), Thane -401 107, Maharashtra.

Year 2019 -2020

Declaration

We declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. We

understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been

properly cited or from whom proper permission has not been taken when needed.

Ashish Roy

Roll No.:46

Priti

Priti Negi

Roll No.:32

Alai Parekh

Roll No.:36

Date: 22/09/2020

Shree Rahul Education Society's (Regd.)
SHREE L. R. TIWARI COLLEGE OF ENGINEERING
Shree L. R. Tiwari Educational Campus, Mira Road (East), Thane 401107, Maharashtra
(Approved by AICTE, Govt. of Maharashtra & Affiliated to University of Mumbai)
NAAC Accredited | ISO 9001:2015 Certified | DTE Code No. 3423
Tel. No.: 022-28120144 / 45 / 42 | Email: slrtce@rahuleducation.com | Website: www.slrtce.in

CERTIFICATE

This is to certify that the project entitled **"Bit-Contracts: Smart Contracts for Cars."** is a bonafide work of

Ashish Roy (46)

Priti Negi (32)

Alai Parekh (36)

submitted to the University of Mumbai in partial fulfilment of the requirements for the award of the degree of **"Bachelor of Engineering"** in **"Computer Engineering"**.

Signature of Supervisor/Guide

Name:Dr. Vinayak D. Shinde

Date: 22/09/2020

Signature of the H.O.D.

Name: Dr. Vinayak D. Shinde

Date: 22/09/2020

Signature of the Principal

Name: Dr. S. Ram Reddy

Date:22/09/2020

Shree Rahul Education Society's (Regd.)
SHREE L. R. TIWARI COLLEGE OF ENGINEERING
Shree L. R. Tiwari Educational Campus, Mira Road (East), Thane 401107, Maharashtra
(Approved by AICTE, Govt. of Maharashtra & Affiliated to University of Mumbai)
NAAC Accredited | ISO 9001:2015 Certified | DTE Code No. 3423
Tel. No.: 022-28120144 / 45 / 42 | Email: slrtce@rahuleducation.com | Website: www.slrtce.in

Project Report Approval

This project report entitled "**Bit-Contracts: Smart Contracts for Cars.**" by **Ashish Roy, Priti Negi, Alai Parekh** is approved for the degree of Bachelor of Engineering in Computer Engineering.

Examiners

1. Name: ____________________________

   Signature: _________________________

2. Name: ____________________________

   Signature: _________________________

Date: 22/09/2020

Place:

Contents

List of Tables

List of Figures

Abstract

Security and Legitimacy are two most vital factors on which a trade is conducted. Usually a third party is involved who acts as a mediator to verify trade between two parties. As a result, the confidentiality and integrity of trade is compromised. Moreover, these transactions have valuable information of the two parties that are stored on traditional database systems which could be jeopardised. We propose an efficient solution for providing a smart contract between the functionality of a car buying and selling system that allows individuals to sell or buy their personal, underused cars in a completely decentralized manner; annulling the necessity of an intermediary. Our answer, named Bit-Contracts, leverages smart contracts and uses them to carry out secure and personal car booking and payments.

Chapter 1

Introduction

1. Introduction

Here we will elaborate the aspects like description, problem statement, scope, motivation and also the objectives of our project

1.1 Description

A blockchain is a digitized, decentralized, public ledger of all crypto currency transaction. Constantly growing as ‘completed’ blocks (the most recent transactions) are recorded and added to it in chronological order, it allows market participants to keep track of digital currency transactions without central recordkeeping. Each node (a computer connected to the network) gets a copy of the block chain, which is downloaded automatically.

Originally developed as the accounting method for the virtual currency Bitcoin, blockchains – which use what's known as distributed ledger technology (DLT) – are appearing in a variety of commercial applications today. Currently, the technology is primarily used to verify transactions, within digital currencies though it is possible to digitize, code and insert practically any document into the blockchain. Doing so creates an indelible record that cannot be changed; furthermore, the record’s authenticity can be verified by the entire community using the blockchain instead of a single centralized authority.

1.2 Problem statement

The topic of our research is smart contract using blockchain. There was a strong need of this type of research for the smart contract so that they can perform their tasks effectively and efficiently. Previously, some of the work had done on some of its related topics but our purpose for this research is to exactly find those factors which help the managers to manage the smart contract better.

- Some problems come at the scheduling stage.
- Some occur on trust and transparency.

1.3 Scope and Motivation

Inordertounderstandblockchain,exploringthemotivation behind its creation is essential to understand its significance. It was initially referred to as “chain of blocks” in Nakamoto’s paper [1]. This technology enables digital cash to be spent directlyfromonepartytoanotherwithoutanyinvolvementofa third party. In broad sense, blockchain is an immutable, append-only linked list of blocks that are chained together in such way that the alteration of a block will alter itsconsecutive blocks. Fig. 1 illustrates a blockchain and how it works. Each block in a blockchain contains transactions, then it is hashed alongside previous block’s hash. In this way, the hashes are linkedtogether,renderinganyalterationofpreviousblocksalso need to change the nextblock.

It may help collecting perfect transactions in detail. In a very short time, the collection will be obvious, simple and sensible. It will help a person to know the actual process done in-between customer to seller. It will be also reduced the cost of collecting the information and collecting procedure will go on smoothly.

- It satisfies the user requirement.
- Be easy to understand by the user and operator.
- Have a good user interface.
- User friendly.

1.4 Project Objectives

The objectives of the systems development and event management are:

1. It provides fault-tolerance, immutability, transparency and full traceability of the stored transaction records, as well as coherent digital representations of physical assets and autonomous transaction executions.
2. It is essential for the stored records to be tamper-proof, while the best case would be if each actor issuing transactions could do that without relying on any centralized third-party intermediary.
3. **Direct dealings with customers**. Smart contracts remove the need for intermediaries and allow for transparent, direct relationships with customers.
4. **More trust**. Business agreements are automatically executed and enforced. Plus, these agreements are immutable and therefore unbreakable.
5. **Record keeping**. All contract transactions are stored in chronological order in the blockchain and can be accessed along with the complete audit trail.

Chapter 2

Literature Review

2. Literature Review

Here we will elaborate the aspects like the literature survey of the project and what all projects are existing and been actually used in the market which the makers of this project took the inspiration from and thus decided to go ahead with the project covering with the problem statement.

Literature Survey Papers:

Sr No	Name Of Paper	Author and Publication year	Abstract
1.	Blockchain: A Distributed Solution to Automotive Security and Privacy	Ali Dorri , Marco Steger , Salil S. Kanhere, Raja Jurdak	➢ Interconnected smart vehicles offer a range of sophisticated services that benefit the vehicle owners, transport authorities, car manufacturers, and other service providers. ➢ This potentially exposes smart vehicles to a range of security and privacy threats such as location tracking or remote hijacking of the vehicle. In this article, we argue that blockchain (BC), a disruptive technology that has found many applications from cryptocurrencies to smart contracts, is a potential solution to these challenges. We propose a BC-based architecture to protect the privacy of users and to increase the security of the vehicular ecosystem. ➢ Wireless remote software updates and other emerging services such as dynamic vehicle insurance fees are used to illustrate the efficacy of the proposed security architecture. We also qualitatively argue the resilience of the architecture against common security attacks.

2.	Blockchain-based Internet of Vehicles: Distributed Network Architecture and Performance Analysis	**Tigang Jiang** **Hua Fang**	➢ The rapid growth of Internet of Vehicles (IoV) has brought huge challenges for large data storage, intelligent management, and information security for the entire system. The traditional centralized management approach for IoV faces the difficulty in dealing with real time response. ➢ The blockchain, as an effective technology for decentralized distributed storage and security management, has already showed great advantages in its application of Bitcoin. In this paper, we investigate how the blockchain technology could be extended to the application of vehicle networking, especially with the consideration of the distributed and secure storage of big data. ➢ We define several types of nodes such as vehicle and roadside for vehicle networks and form several sub-blockchain networks. In the paper, we present a model of the outward transmission of vehicle blockchain data, and then give detail theoretical analysis and numerical results. Our study has shown the potential to guide the application of Blockchain for future vehicle networking.
3.	Blockchain and Its Coming Impact on Financial Services	Kurt Fanning Search for more papers by this author David P. Centers	➢ This article provides insight into how “Blockchains” work. A block chain or Blockchain is a distributed database that maintains a continuously growing list of data records that are hardened against tampering and revision, even by operators of the data store's nodes. ➢ One can view a Blockchain as a public ledger of all transactions that have ever been executed. It is constantly growing as completed blocks are added to previous blocks forming a chain. Importantly, blocks are added to the Blockchain in a linear, chronological

			order. Each miner gets a copy of the Blockchain when joining the Bitcoin network. ➢ The Blockchain they receive has complete and accurate information about the addresses and their balances right from the genesis block to the most recently completed block. © 2016 Wiley Periodicals, Inc.
4.	Blockchains Can Work for Car Insurance: Using Smart Contracts and Sensors to Provide On-Demand Coverage	Fabrizio Lamberti , Valentina Gatteschi, Claudio Demartini, Matteo Pelissier, Alfonso Gomez, Victor Santamaria	➢ Blockchains and sensors installed on a vehicle could be combined to semi automatically activate/deactivate car insurance coverage in an envisaged on-demand insurance scenario. ➢ We present a prototype that includes a mobile application (app) and a portable electronic device to be installed onboard. The mobile app lets the driver dynamically change the status of specific insurance coverage (in some cases, after pictures of the vehicle have been taken to attest to its conditions). ➢ Each modification and picture hash (a fixed-length alphanumeric summary of data content) are saved on the blockchain within a smart contract to certify changes made as well as the vehicle's status. Sensors embedded in the electronic device are used to collect passengers' and the vehicles data. Data are then used to automatically modify insurance coverage based on car/environment conditions and the preferences set. ➢ The proposed solution could help lower policy modification costs and limit insurance fraud.

5.	TheMultimedia Blockchain: A Distributed and Tamper-Proof Media Transaction Framework	Deepayan Bhowmik∗ and Tian Feng† ∗Department of Computing, Sheffield Hallam University, Sheffield, United Kingdom, S1 1WB †Dept. of Electrical & Electronic Engineering, The University of Sheffield, Sheffield, United Kingdom, S1 4DE deepayan.bhowmik@shu.ac.uk and t.feng@sheffield.ac.uk	➢ A distributed and tamper proof media transaction framework is proposed based on the blockchain model. ➢ Current multimedia distribution does not preserve self-retrievable information of transaction trails or content modification histories. For example, digital copies of valuable artworks, creative media and entertainment contents are distributed for various purposes including exhibitions, gallery collections or in media production workflow. ➢ Original media is often edited for creative content preparation or tampered with to fabricate false propaganda over social media. However, there is no existing trusted mechanism that can easily retrieve either the transaction trails or the modification histories. We propose a novel watermarking-based Multimedia Blockchain framework that can address such issues. The unique watermark information contains two pieces of information: a) a cryptographic hash that contains transaction histories (blockchain transactions log) and b) an image hash that preserves retrievable original media content. Once the watermark is extracted, first part of the watermark is passed to a distributed ledger to retrieve the historical transaction trail and the latter part is used to identify the edited / tampered regions. ➢ The paper outlines the requirements, the challenges and demonstrates the proof of this concept.

2.1 Proposed System

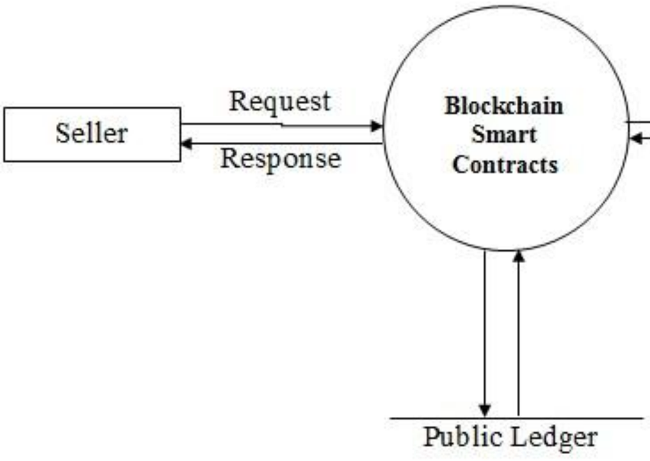


Figure 1: System Model of Bit-Contracts

As illustrated in Figure 1, our system model consists of following entities:

- Seller: An individual with the intent to sell his/her car.
- Buyer: An individual in need of a car.
- Car: A car that is available for trade. Car’s access provision could be handled by any robust access provision protocols like SePCAR .
- Public Ledger: A record where all transaction details have been stored that maintains participant’s identities in secure and anonymous form with their respective cryptocurrency balances.
- Smart Contract: This smart contract is accountable for receiving a booking request, ensuring that acceptable deposits are created by the buyer, handling cancellation requests, dealing with fraudulent activities, and a smooth procedure in general.

Chapter 3

Requirements Analysis and Planning

3. Requirement Analysis and Planning

In requirements analysis encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product or project, taking account of the possibly conflicting requirement s of the various stakeholder s,analysing, documenting, validating and managing software or system requirements. Project planning is part which relates to the use of schedules such as Gantt charts to plan and subsequently report progress within the project environment. Initially, the project scope is defined and the appropriate methods for completing the project are determined

3.1 Functional Requirements

3.1.1 User login

For checking authenticate user, System will provide Login form if it will not register yet then user can register using the registration form.

3.1.2 Transfer Money option

User will transfer money according to your purchase. For transferring money to the seller, system will convert the money into bit currency.

3.1.3 Payment option

After short-listing the car user can give payment accordingly. For payment user must require sufficient amount in their respective account.

3.1.4 Notification of the car

Suppose if the user is donewith the payment, a confirmation message will pop up for the user.

3.1.5 Business Rules

According to business rule payment will be taken according to user requirement of car, so that all users have proportional bill amount.

3.2 Non-functional requirements

3.2.1 Efficiency

As per user need this system evolve quickly that means it requires less time to give feedback. for better performance and throughput, it will have less feedback time so that efficiency can increase.

3.2.2 Reliability

The reliability of our system is predicted to be very high and critical for proper functioning of the overall system. Our reliability is a factor which will be focused more than any other factors affecting it. System work as per the input or user requirement where there will be almost negligible error rate.

3.2.3 Scalability.

If your platform is tie to a platform like Blockchain implementation such as Ethereum then it is also tied to the scalability.

3.2.4 Security

System gives access to that user which are authenticated that means no modification will be done.

3.3 System requirements

This section contains all of the functional and quality requirements of the system. It gives a detailed description of the system and all its features.

3.3.1 Hardware requirements

- **Processor: Pentium 4 and more:**

Pentium 4 processors have an integrated heat spreader (IHS) that prevents the die from accidentally being damaged when mounting and unmounting cooling solutions.

- **RAM: 4GB or more**

Random-access memory (RAM) is a form of computer memory that can be read and changed in any order, typically used to store working data and machine code. A random-access memory device allows data items to be read or written in almost the same amount of time irrespective of the physical location of data inside the memory

- **Hard Disk: 16GB or more**

A hard disk drive (HDD), hard disk, hard drive, or fixed disk[b] is an electro-mechanical data storage device that uses magnetic storage to store and retrieve digital data using one or more rigid rapidly rotating platters coated with magnetic material

3.3.2 Software requirements

○ **Operating System: Windows2000/XP/7/8/10**

An operatingsystem (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs. Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources.

○ **Frontend: Java (Jsp/Servlet)**

Java is a general-purpose programming language that is class-based, object-oriented, and designed to have as few implementation dependencies as possible. It is intended to let application developers *write once, run anywhere* (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

○ **Backend: MySQL**

MySQL is free and open-source software under the terms of the GNU General Public License, and is also available under a variety of proprietary licenses.

○**Eclipse:**

Eclipse is an integrated development environment (IDE) used in computer programming. It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages via plug-ins.

3.4 Project Planning

3.4.1 Timeline Project

The following tables give the project plan for phase 1 and 2 of the project:

PHASE 1

Table 3.1 Project Timeline Phase -1

Activity	**Description**	**Effort in person week**	**Deliverable**

P-01	Group and Domain selection	1 week	Requirement Gathering
P-02	Project Selection	1 week	Existing System Study & Literature Survey
P-03	Project Proposal	1 week	Feasibility Study
P-04	Literature Review	2 weeks	Report
P-05	Software Requirement Specifications	2 weeks	Module Description
P-06	Design Documentation and Implementation	4 weeks	Implementation of important modules.
P-07	Final Report and Log Report	2 weeks	Weekly update and final report
	Total	12 weeks	

PHASE 2

Table 3.2 Project Timeline Phase -2

Activity	**Description**	**Effort in person week**	**Deliverable**
P-01	Identify Hardware and make connections.	3	Connection of every hardware component with NodeMCU
P-02	Connecting Database and Creating rest API	2	Creating database on MySQL and Making of api using hibernate.
P-03	Front end and consumption of api	2	Consuming Restful Api using HTTP protocol

P-04	Build Payment functionality	1	Using Paytm gateway.
P-05	Connecting backend and NodeMCU with cloud	3	Using MQTT protocol we connect.
P-06	Testing	1	Module wise testing
P-07	Deploying	1	Hosting on cloud platform
P-08	Model Making	1	We collect Raw material for making layout of our system
	Total	14 weeks	

3.4.2 Task distribution

GROUP MEMBERS NAME	POSITION OF EVERY MEMBER	TASK PERFORMED BY MEMBER
Ashish Roy	Team Leader	Analysis, Designing, Documentation and Coding.
Priti Negi	Team Member	Analysis, Coding, Designing, Testing, Documentation
Alai Parekh	Team Member	Concept, Documentation, Testing, Coding

Table 3.3 Task Distribution

3.5 Cost of Transaction and Deployment

Table 1: Cost of Deployment

Deployment Cost in Gas	Deployment Cost in INR
5,36,467	110.06

Table 2: Cost of Transaction

Transaction	Cost in Gas	Cost in INR

Store Encrypted details in Smart Contracts	64,458	13.22
Request for the Car	28,685	5.88
Cancellation	43,752	8.98
Triggering the distribution of earnings	30,874	6.33
Withdraw Money from Ether wallet	25,970	5.33

*These values are an approximate estimation based on the conversion rate of 1 ether = 10,456.52 INR [10] and may vary depending on the exchangerate.

3.5.1Gas

Gas is a unit that measures the amount of computational effort that it will take to execute certain operations. Every single operation that takes part in Ethereum, be it a simple transaction, or a smart contract, or even an ICO takes some amount of gas [11] [2]. Gas is what is used to calculate the amount of fees that need to be paid to the network in order to execute an operation. Miners get paid an amount in Ether which is equivalent to the total amount of gas it took them to execute a completeoperation.

3.5.2DeploymentCost

All transactions in Ethereum cost 21000 gas as a base. Which means that if we are just transferring funds and not signing with a contract, your transaction takes 21000 gas. If you are signing with a contract, your transaction takes 21000 gas plus any gas associated with running the contract.

On combining the following parameters, the main cost of deploying the smart contracts can be calculated.

- The costs related to storing the contract code (200 gas perbyte).
- Cost of storing additional data on the contract (20,000 gas per 256-bitword).
- 32,000 gas have to be paid to create a new transaction (deployment of a brand-new contract) whereas the base cost of each transaction is 21,000gas.

To calculate the deployment costs of our smart contracts, we have used Remix IDE [12]. The actual deployment cost varies by the size of the smart contract and the total bytes in its storage.

3.5.3Transaction Cost

The cost for executing each transaction cost can be calculated as follows.

- Minimum cost of each transaction (21,000gas)
- Cost when 256-bit word is stored on the smart contract (20,000gas)
- Cost of storing additional data on the contract (5,000gas)
- Making a decision on transaction having a financial value (9,000gas)

Chapter 4

Analysis Modelling

4. Analysis Modelling

In this chapter, all the aspects of the proposed system will be covered in the diagrammatic manner and provides the detailed manner of the system.

4.1 BehaviouralModelling

UML behavioural diagrams visualize, specify, construct, and document the dynamic aspects of a system. The behavioural diagrams are categorized as follows: use case diagrams, interaction diagrams, state–chart diagrams, and activity diagrams.

4.1.1 Use Case Diagram

In the Figure 4.1 of use case diagram, System consists of two actors which are buyer and seller.User consist of three use cases which can be accessible only if user is authenticated, hence for that operation user needs to login. After that user can use other functionality which is select car option, view selected carand confirm the payment.

After completion of total purchase, then the system will notify user. also, it will continuously notify user about their purchase.

Other actor is seller in this system, which can have a full report of purchase from particular user. All these features are controlled by the bit contract which is nothing but based on Blockchain.

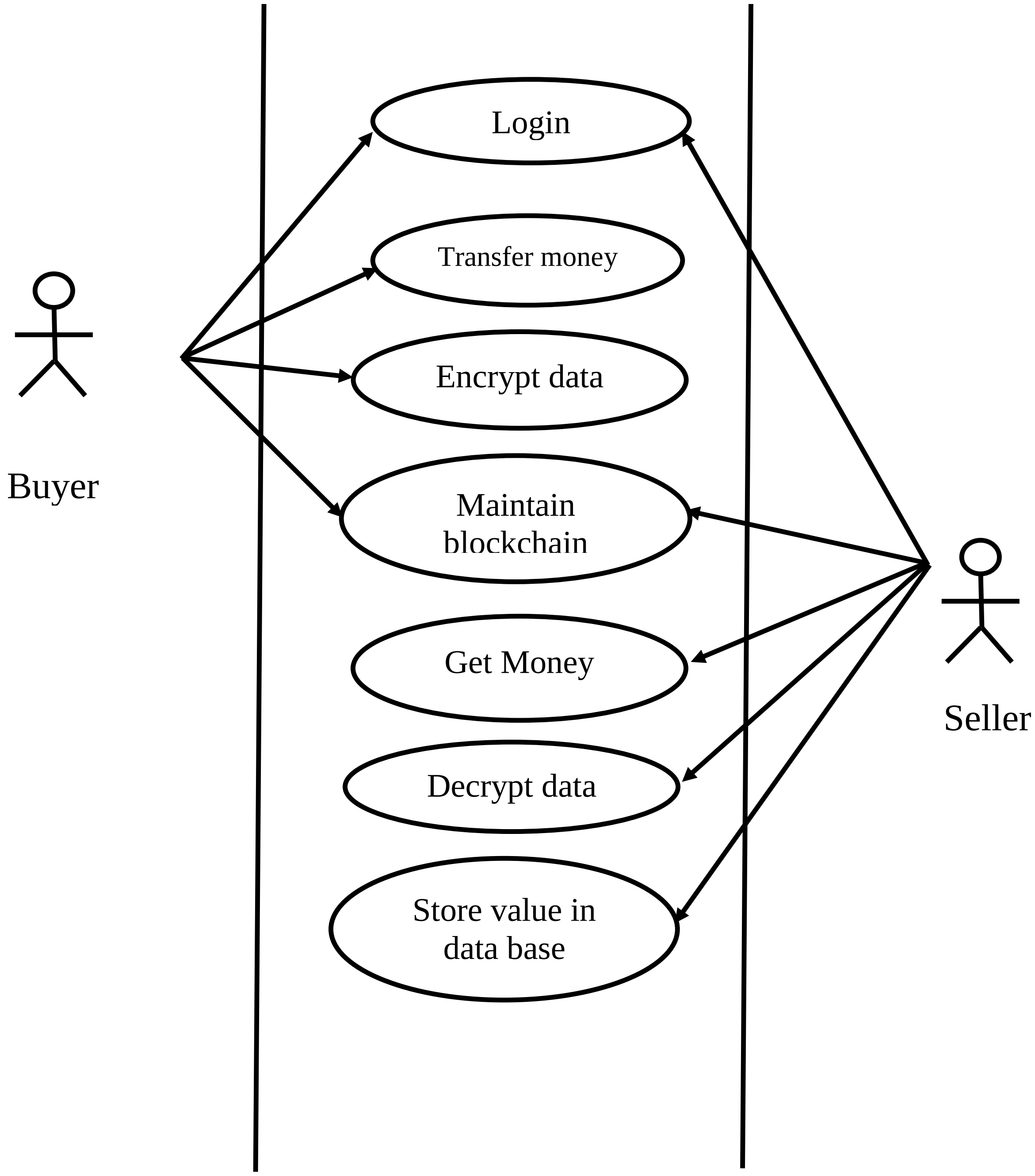
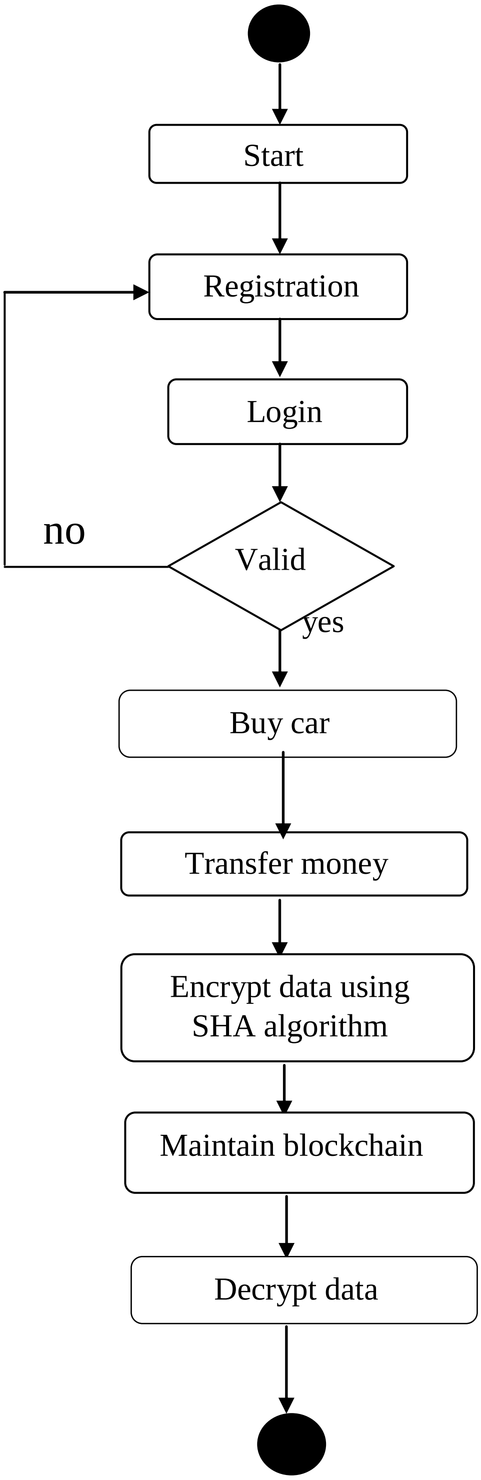


Figure 4.1 Use Case diagram

4.1.2 Activity Diagram

Buyer activity:



Seller activity:

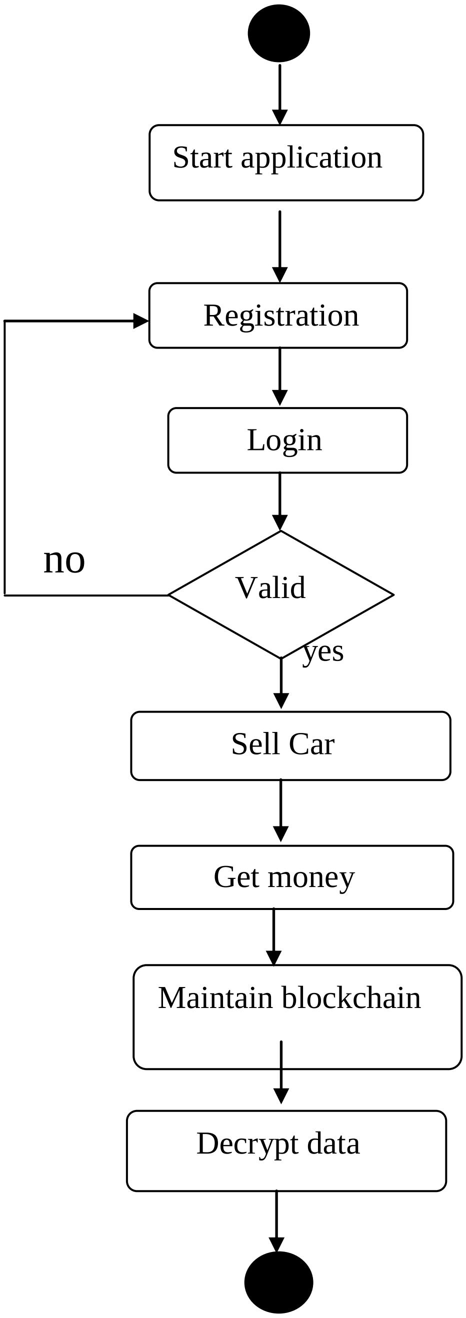


Figure 4.3 Activity Diagram of Seller

4.2 Data Flow Diagram

A data flow diagram (DFD) is a graphical representation of the flow of data through an information system. A data flow diagram can also be used for the visualization of data processing (structured design). It is common practice for a designer to draw a context-level DFD first which shows the interaction between the system and outside entities. This context-level DFD is then exploded to show more detail of the system being modelled.

4.2.1 Level 0 of DFD

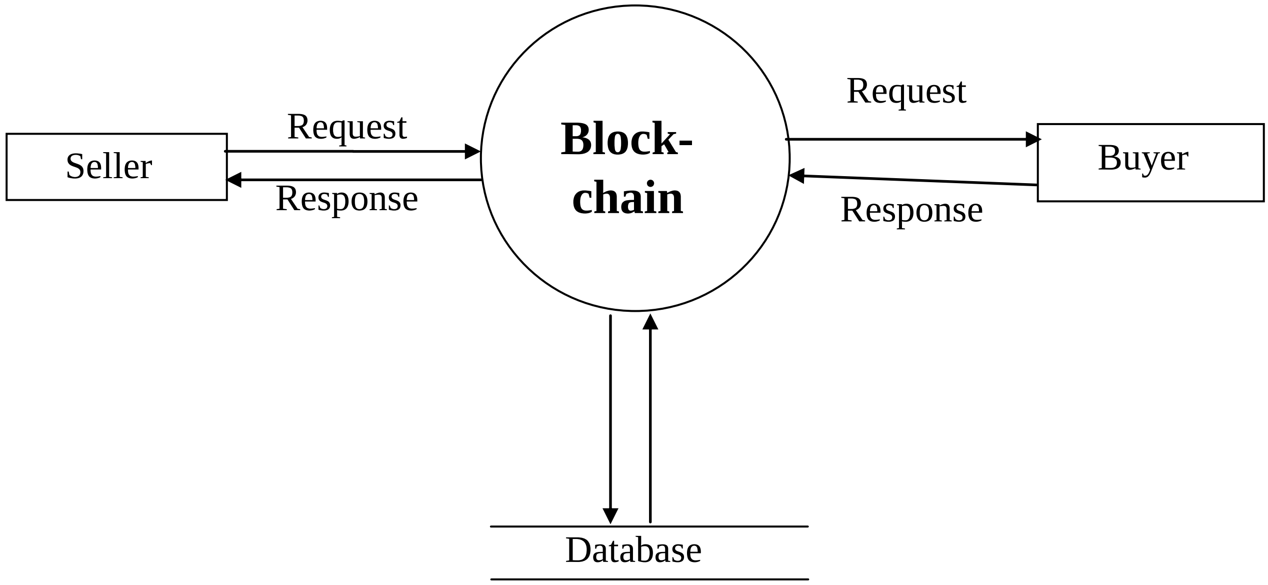
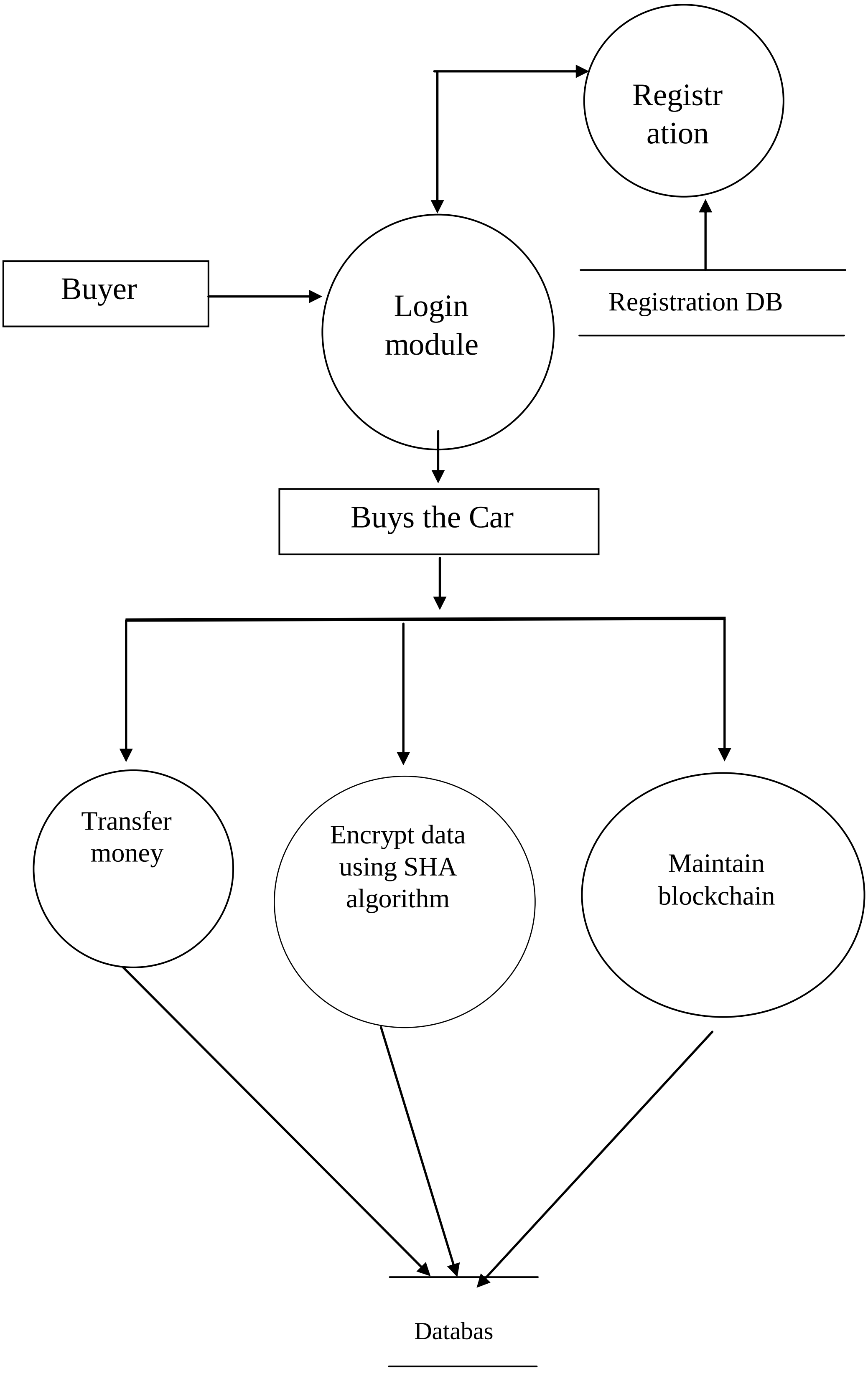
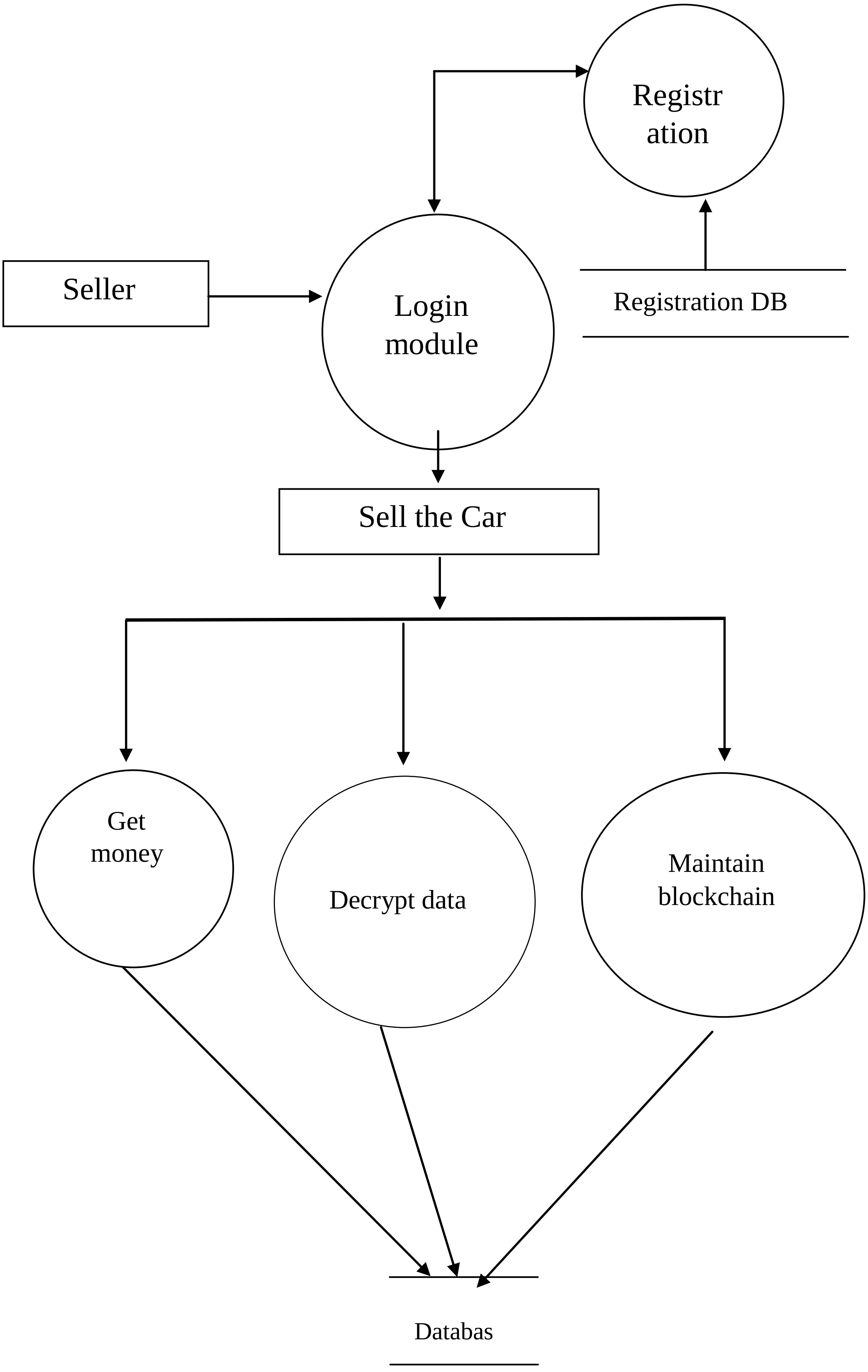


Figure 4.5 DFD level 0 of System

4.2.2 DFD for Buyer



4.2.3 DFD for Seller:



4.3 Database Design

4.3.1 ER Diagram

Figure 4.8 represent Entity relationship diagram, is a graphical representation that depicts relationships among people, objects, places, concepts. In our project, we have created 2 entities i.e. buyer and seller and each entity have its own attributes and primary key and entities has relations between them.

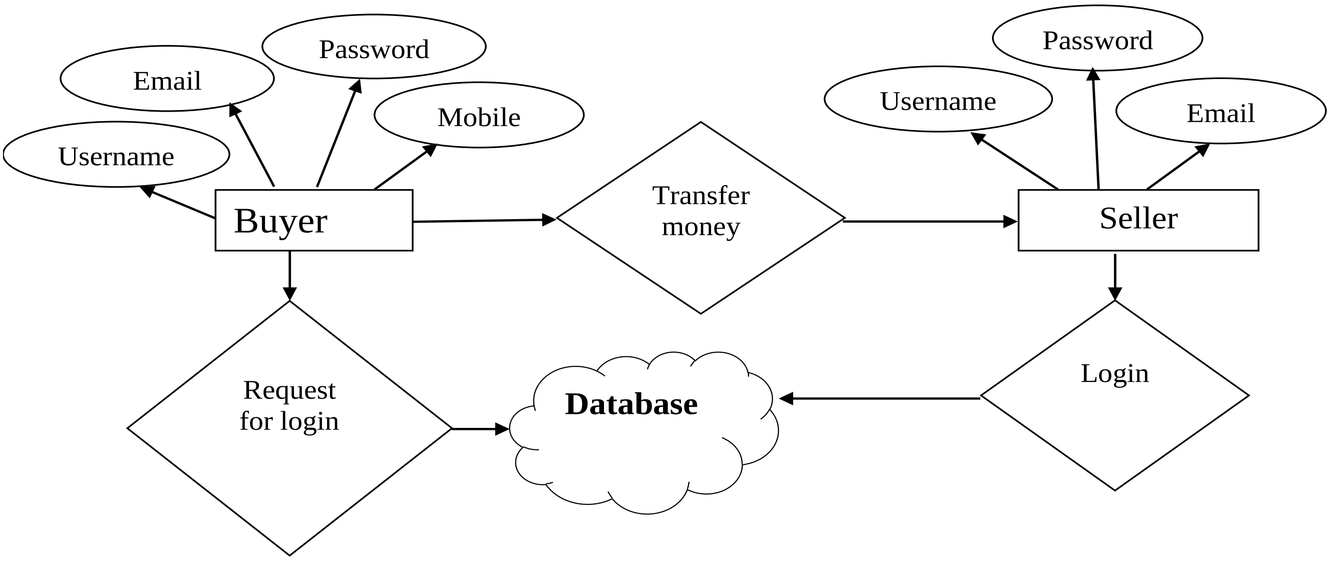


Figure 4.3 ER Diagram

Chapter 5

System Design

5. System Design

Design will elaborate the process of describing, organising and structuring the components of the system both at the architectural level and at the detailed level.

5.1 System Flow

Figure 5.1 illustrates the system flow of Bit-Contracts.

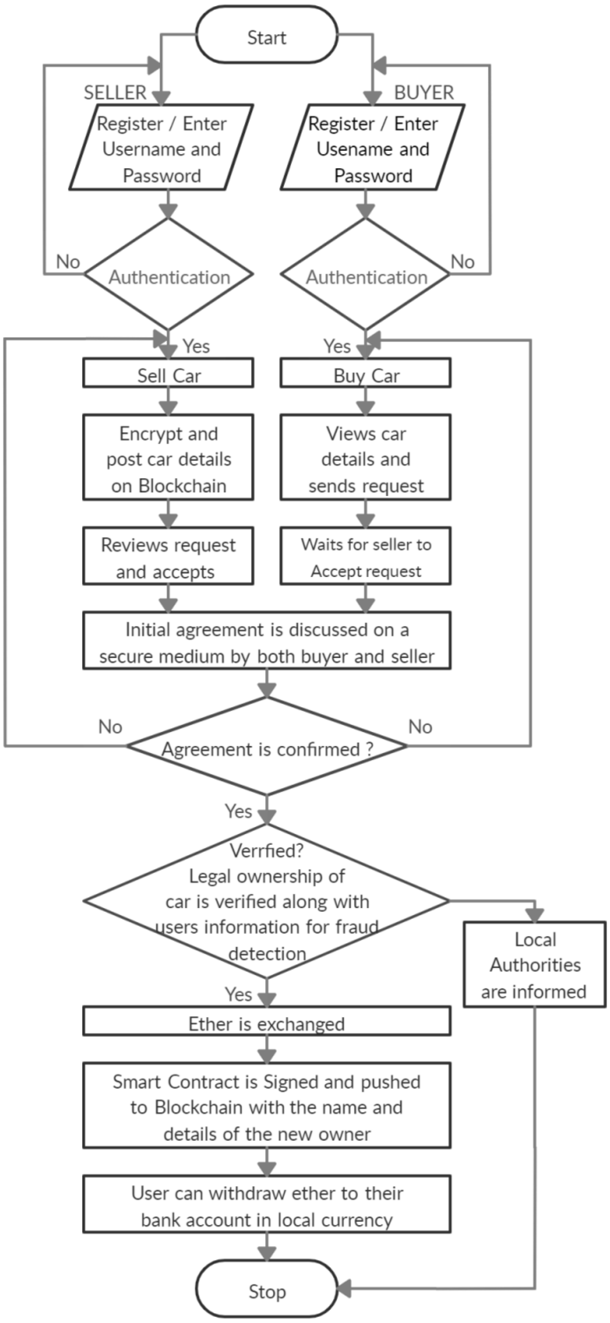

Start
SELLER
BUYER
Register / Enter Username and Password
Register / Enter Usename and Password
Authentication
Authentication
No
No
Yes
Yes
Sell Car
Buy Car
Encrypt and post car details on Blockchain
Views car details and sends request
Reviews request and accepts
Waits for seller to Accept request
Initial agreement is discussed on a secure medium by both buyer and seller
No
No
Agreement is confirmed ?
Yes
Verrfied? Legal ownership of car is verified along with users information for fraud detection
Local Authorities are informed
Yes
Ether is exchanged
Smart Contract is Signed and pushed to Blockchain with the name and details of the new owner
User can withdraw ether to their bank account in local currency
Stop

Figure 5.1 System Flow

5.2 Class Diagram

The class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code
of the software application.

The class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modelling of object-oriented systems because they are the only UML diagrams which
can be mapped directly with object-oriented languages.

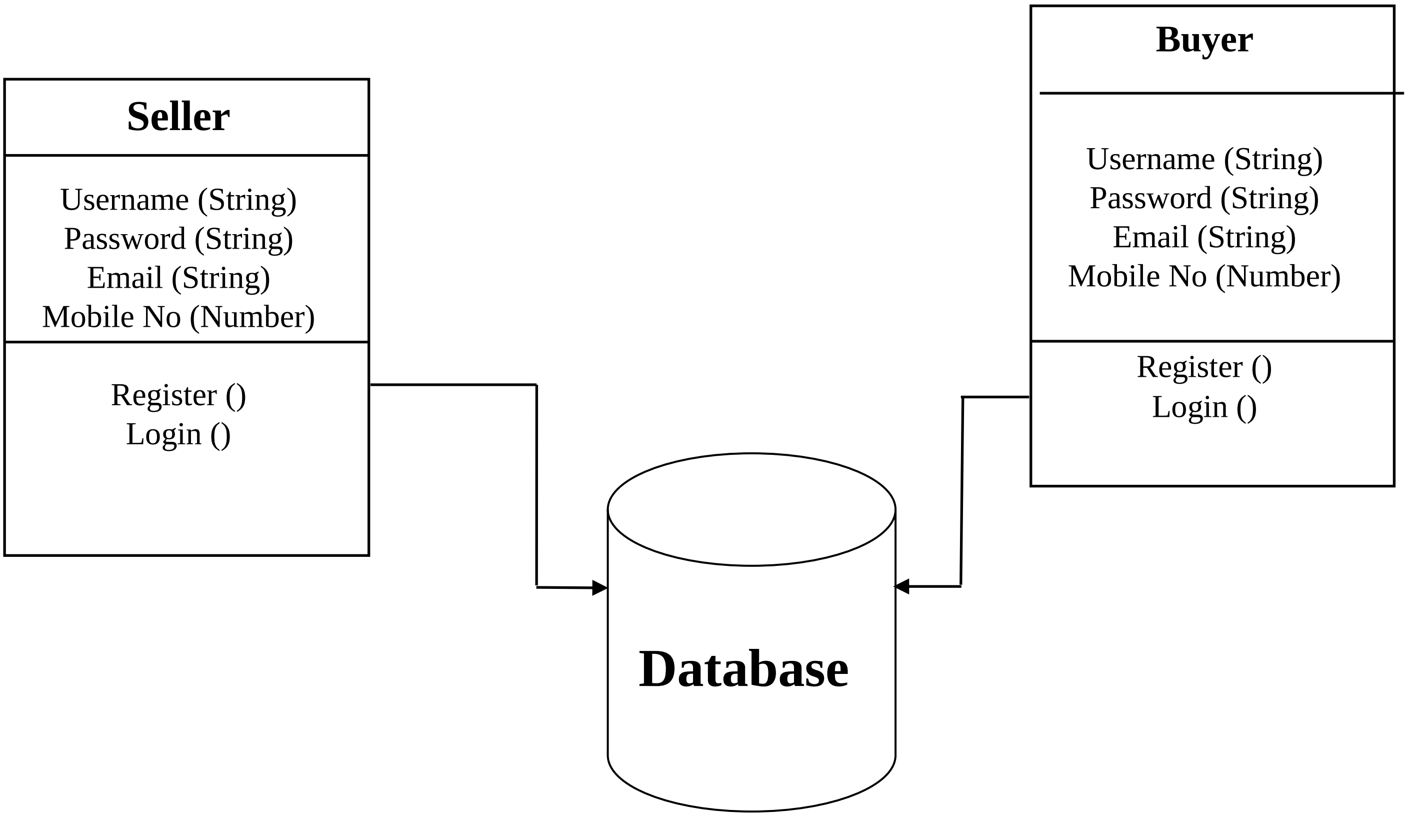


Fig.5.2 Class Diagram

5.3 Graphical User Interface

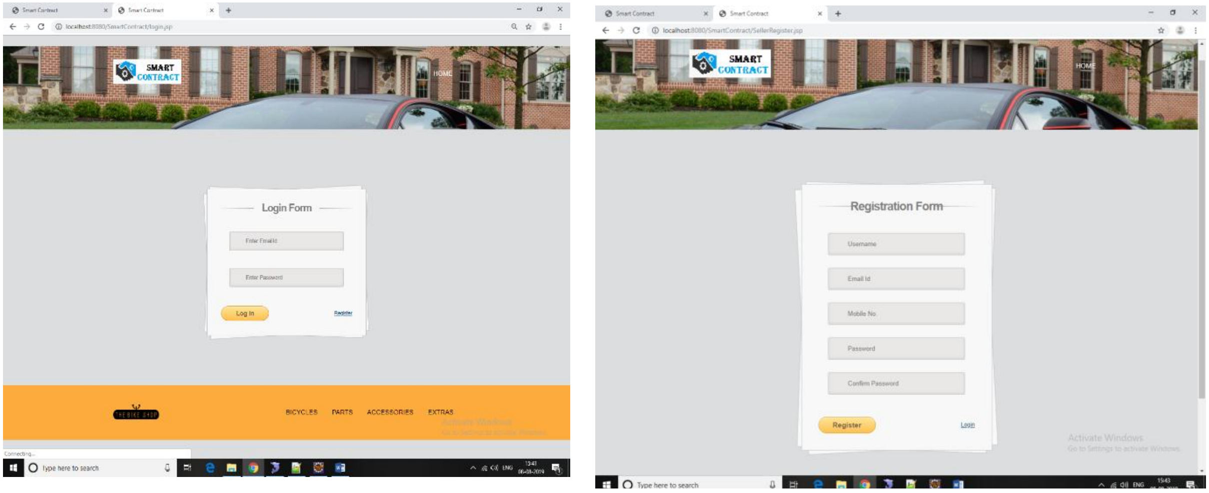


Fig 5.3 Web Pages

Figure 5.6(a)is Login page for user authentication. Login page required Username and password.

Figure 5.6(b) this is registration form page, where if user is not registered on our system then by using this form user can register.

Chapter 6

Implementation

6. Implementation

6.1 Main Page Code

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<%@ include file="header1.jsp"%>
<%@ page import="Database.DatabaseConnection" %>
<%@ page import="java.sql.ResultSet"%>

<%
DatabaseConnection db = new DatabaseConnection();
db.dbconnection();
//int bid =(Integer) session.getAttribute("bid");
String query =  "SELECT * FROM cardetails";
System.out.println(query);
ResultSet rs = db.getResultSet(query);
%>

<div class="bikes">
<div class="mountain-sec">
	<h2>VEHICLES</h2>
	<%
		while(rs.next())
		{
			int pid = rs.getInt("id");
			String brand = rs.getString("brand");
			String imagename = rs.getString("imagename");
	 %>
	<a href="single.jsp?productid=<%=pid%>"><div class="bike">
		<img src="image123/<%=imagename%>" alt="<%=imagename%>"/>
	<div class="bike-cost">
				<div class="bike-mdl">
					<h4>NAME<span><%=brand %></span></h4>
				</div>
				<div class="bike-cart">
					<a   class="buy"   href="single.jsp?productid=<%=pid%>">BUY
NOW</a>
				</div>
				<div class="clearfix"></div>
			</div>
			<!-- <div class="fast-viw">
					<a href="single.html">Quick View</a>
			</div> -->
		</div></a>
	<%}%>

		<div class="clearfix"></div>
</div>
```

```
</div>
```

6.2 Index Code

```
<!DOCTYPE html>
<html>
<head>
<title>Smart Contract</title>
<link href="css/bootstrap.css" rel='stylesheet' type='text/css' />
<!-- jQuery (Bootstrap's JavaScript plugins) -->
<script src="js/jquery.min.js"></script>
<!-- Custom Theme files -->
<link href="css/style.css" rel="stylesheet" type="text/css" media="all" />
<!-- Custom Theme files -->
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta name="keywords" content="Bike-shop Responsive web template, Bootstrap Web Templates, Flat Web Templates, Andriod Compatible web template,
Smartphone Compatible web template, free webdesigns for Nokia, Samsung, LG, SonyErricsson, Motorola web design" />
<script type="application/x-javascript"> addEventListener("load", function() { setTimeout(hideURLbar, 0); }, false); function hideURLbar(){ window.scrollTo(0,1); } </script>
<!--webfont-->
<link href='http://fonts.googleapis.com/css?family=Roboto:500,900,100,300,700,400' rel='stylesheet' type='text/css'>
<!--webfont-->
<!-- dropdown -->
<script src="js/jquery.easydropdown.js"></script>
<link href="css/nav.css" rel="stylesheet" type="text/css" media="all"/>
<script src="js/scripts.js" type="text/javascript"></script>
<!--js-->
<!---- start-smoth-scrolling---->
		<script type="text/javascript" src="js/move-top.js"></script>
		<script type="text/javascript" src="js/easing.js"></script>
		<script type="text/javascript">
			jQuery(document).ready(function($) {
				$(".scroll").click(function(event){
					event.preventDefault();
					$('html,body').animate({scrollTop:$(this.hash).offset().top},900);
				});
			});
		</script>

<!---- start-smoth-scrolling---->


</head>
<body>
<!--banner-->
<script src="js/responsiveslides.min.js"></script>
```

```
<script>
   $(function () {
     $("#slider").responsiveSlides({
auto: true,
nav: true,
speed: 500,
      namespace: "callbacks",
      pager: true,
     });
   });
</script>
<div class="banner-bg banner-bg1">
<div class="container">
        <div class="header">
        <div class="logo">
                    <a href="index.jsp"><img src="images/logo.png" alt=""/></a>
            </div>
            <div class="top-nav">

                    <label class="mobile_menu" for="mobile_menu">
                    <span>Menu</span>
                    </label>
                    <input id="mobile_menu" type="checkbox">
                <ul class="nav">

                    <li class="dropdown1"><a href="buyerlogin.jsp">Login</a>

                    </li>



                </ul>
            </div>
            <div class="clearfix"></div>
        </div>
</div>
<div class="caption">
    <div class="slider">
                <div class="callbacks_container">
                    <ul class="rslides" id="slider">
                        <li><h1>HANDMADE BICYCLE</h1></li>
                            <li><h1>SPEED BICYCLE</h1></li>
                            <li><h1>MOUINTAIN BICYCLE</h1></li>
                    </ul>
                    <p>You <span>create</span> the <span>journey,</span> we supply the <span>parts</span></p>

                </div>
            </div>
</div>
<div class="dwn">
    <a class="scroll" href="#cate"><img src="images/scroll.png" alt=""/></a>
```

```
</div>
</div>

<!--bikes-->
<div class="bikes">
        <h3>POPULAR BIKES</h3>
        <div class="bikes-grids">
                <ul id="flexiselDemo1">
                        <li>
                                <img src="images/bik1.jpg" alt=""/>
                                <div class="bike-info">
                                        <div class="clearfix"></div>
                                </div>

                        </li>
                        <li>
                                <img src="images/bik2.jpg" alt=""/>
                                <div class="bike-info">
                                        <div class="clearfix"></div>
                                </div>
                        </li>
                        <li>
                                <img src="images/bik3.jpg" alt=""/>
                                <div class="bike-info">
                                        <div class="clearfix"></div>
                                </div>

                        </li>
                        <li>
                        <img src="images/bik4.jpg" alt=""/>
                                <div class="bike-info">

                                        <div class="clearfix"></div>
                                </div>

                        </li>
                        <li>
                                <img src="images/bik5.jpg" alt=""/>
                                <div class="bike-info">
                                        <div class="clearfix"></div>
                                </div>

                        </li>
                        <li>
                        <img src="images/bik6.jpg" alt=""/>
                                <div class="bike-info">
                                        <div class="clearfix"></div>
                                </div>

                        </li>
        </ul>
```

```
            <script type="text/javascript">
             $(window).load(function() {
              $("#flexiselDemo1").flexisel({
                    visibleItems: 3,
                    animationSpeed: 1000,
                    autoPlay: true,
                    autoPlaySpeed: 3000,
                    pauseOnHover:true,
                    enableResponsiveBreakpoints: true,
                    responsiveBreakpoints: {
                        portrait: {
                            changePoint:480,
                            visibleItems: 1
                        },
                        landscape: {
                            changePoint:640,
                            visibleItems: 2
                        },
                        tablet: {
                            changePoint:768,
                            visibleItems: 3
                        }
                    }
            });
            });
            </script>
            <script type="text/javascript" src="js/jquery.flexisel.js"></script>
</div>
</div>
<!---->
<div class="contact">
<div class="container">
    <h3>CONTACT US</h3>
    <p>Please contact us for all inquiries and purchase options.</p>
    <form>
        <input type="text" placeholder="NAME" required="">
        <input type="text" placeholder="SURNAME" required="">
        <input class="user" type="text" placeholder="USER@DOMAIN.COM"
required=""><br>
        <textarea placeholder="MESSAGE"></textarea>
        <input type="submit" value="SEND">
    </form>
</div>
</div>
<!---->
<div class="footer">
<div class="container wrap">
    <div class="logo2">
        <a href="#"><img src="images/logo2.png" alt=""/></a>
    </div>
    <div class="ftr-menu">
```

```
                <ul>
                        <li><a href="#">BICYCLES</a></li>
                        <li><a href="#">PARTS</a></li>
                        <li><a href="#">ACCESSORIES</a></li>
                        <li><a href="#">EXTRAS</a></li>
                </ul>
        </div>
        <div class="clearfix"></div>
</div>
</div>
<!---->

</body>
</html>
```

6.3 Buyer Register Code

```
package Authentication;

import java.io.IOException;
import java.io.PrintWriter;
import java.security.InvalidAlgorithmParameterException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.web3j.crypto.CipherException;

import Database.DatabaseConnection;
import blockchainevoting.creatingaccount;

/**
 * Servlet implementation class BuyerRegister
 */
@WebServlet("/BuyerRegister")
public class BuyerRegister extends HttpServlet {
private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public BuyerRegister() {
        super();
        // TODO Auto-generated constructor stub
    }
```

```
/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub

        PrintWriter out = response.getWriter();
      String name= request.getParameter("name");
        String email= request.getParameter("email");
        String mobile= request.getParameter("mobile");
        String address= request.getParameter("address");
        String pass1= request.getParameter("pass1");

      DatabaseConnection db = new DatabaseConnection();
        db.dbconnection();

        String allaccount="";
        try {
                allaccount = creatingaccount.createblockchainaccount(pass1);
        }    catch    (NoSuchAlgorithmException    |    NoSuchProviderException    |
InvalidAlgorithmParameterException
                        | CipherException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        String accountalldet[]=allaccount.split("@SP");
String account="0x"+accountalldet[0];
String keystorepath=accountalldet[1];



      String queryText = "insert into buyer(name,email,phone,password,address,accountid,path)
values('"+name+"','"+email+"','"+mobile+"','"+pass1+"','"+address+"','"+account+"','"+keystorepath+"')
";
      int i = db.getUpdate(queryText);
      System.out.println(i);
      if(i>0)
        {
                out.println("<script type=\"text/javascript\">");
                out.println("alert('Data inserted Successfully')");
                out.println("location=\"Buyer/buyerlogin.jsp\"");
                out.println("</script>");
        }
}
}
```

6.4 Seller Register Code

```
package Authentication;

import java.io.IOException;
import java.io.PrintWriter;
import java.security.InvalidAlgorithmParameterException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.web3j.crypto.CipherException;

import Database.DatabaseConnection;
import blockchainevoting.creatingaccount;

/**
 * Servlet implementation class SellerRegister
 */
@WebServlet("/SellerRegister")
public class SellerRegister extends HttpServlet {
private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public SellerRegister() {
        super();
        // TODO Auto-generated constructor stub
    }

/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        PrintWriter out = response.getWriter();
     String name= request.getParameter("name");
        String email= request.getParameter("email");
        String mobile= request.getParameter("mobile");
        String pass1= request.getParameter("pass1");

     DatabaseConnection db = new DatabaseConnection();
        db.dbconnection();

        String allaccount="";
        try {
                allaccount = creatingaccount.createblockchainaccount(pass1);
        } catch (NoSuchAlgorithmException | NoSuchProviderException |
InvalidAlgorithmParameterException
                        | CipherException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        String accountalldet[]=allaccount.split("@SP");
String account="0x"+accountalldet[0];
String keystorepath=accountalldet[1];


     String queryText = "insert into admininfo(name,Emailid,phonenumber,password,accountid,path)
values('"+name+"','"+email+"','"+mobile+"','"+pass1+"','"+account+"','"+keystorepath+"') ";
     int i = db.getUpdate(queryText);
     System.out.println(i);
     if(i>0)
        {
                out.println("<script type=\"text/javascript\">");
                out.println("alert('Data inserted Successfully')");
                out.println("location=\"login.jsp\"");
                out.println("</script>");
        }
}
}
```

6.5 Login Code

```
package Authentication;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import Database.DatabaseConnection;

/**
 * Servlet implementation class BuyerLogin
 */
@WebServlet("/BuyerLogin")
public class BuyerLogin extends HttpServlet {
private static final long serialVersionUID = 1L;

   /**
    * @see HttpServlet#HttpServlet()
    */
   public BuyerLogin() {
      super();
      // TODO Auto-generated constructor stub
   }

/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub

        HttpSession session=request.getSession();
        PrintWriter out = response.getWriter();
      String userid= request.getParameter("uname");
```

```
        String password= request.getParameter("pass");

        int id;
     DatabaseConnection db = new DatabaseConnection();
        db.dbconnection();

     String queryText ="select * from buyer where email='"+userid+"' and password='"+password+"'";
     ResultSet rs = db.getResultSet(queryText);
     try
     {
                if(rs.next())
                {
                        String accountid= rs.getString("accountid");
                        System.out.println(accountid);
                        id = rs.getInt("id");
                        //accountid= rs.getInt("id");
                        System.out.println(id);
                        session.setAttribute("bid",id);
                        session.setAttribute("bname",userid);
                        session.setAttribute("accountid",accountid);
                out.println("<script type=\"text/javascript\">");
        out.println("alert('succsefully login')");
        out.println("location=\"Buyer/mainPage.jsp\"");
        out.println("</script>");
                }
                else
                {
        out.println("<script type=\"text/javascript\">");
        out.println("alert('Sorry username or password Invalid')");
        out.println("location=\"Buyer/buyerlogin.jsp\"");
        out.println("</script>");
                }
        } catch (SQLException e) {
                e.printStackTrace();
        }

  }
}
```

6.6 Get Balance Code

```
package blockchainevoting;

import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.concurrent.ExecutionException;

import org.web3j.protocol.Web3j;
import org.web3j.protocol.core.DefaultBlockParameterName;
import org.web3j.protocol.core.methods.response.EthGetBalance;
import org.web3j.protocol.http.HttpService;
import org.web3j.utils.Convert;
import org.web3j.utils.Convert.Unit;

public class gettingBalance {

public static void main(String[] args) throws InterruptedException, ExecutionException {
	// TODO Auto-generated method stub
	// connect to node
	Web3j web3 = Web3j.build(new HttpService("http://localhost:8545")); // defaults to
http://localhost:8545/

	// send asynchronous requests to get balance

	String accountnum="0xf89ef6d367435d85b77dd7a032cb319c7116473a";

	EthGetBalance ethGetBalance = web3
	  .ethGetBalance(accountnum, DefaultBlockParameterName.LATEST)
	  .sendAsync()
	  .get();
	//EthGetBalance balanceWei = web3.ethGetBalance(accountnum,
DefaultBlockParameterName.LATEST).send();
	BigInteger wei = ethGetBalance.getBalance();

	System.out.println("balance is wei "+wei);
	BigDecimal balanceInEther = Convert.fromWei(wei.toString(), Unit.ETHER);
	System.out.println("balance is decimal  "+balanceInEther);

}
public static String getbalanceofuser(String userid) throws InterruptedException, ExecutionException
{
	Web3j web3 = Web3j.build(new HttpService("http://localhost:8545")); // defaults to
http://localhost:8545/

	// send asynchronous requests to get balance

	String accountnum=userid;

	EthGetBalance ethGetBalance = web3
	  .ethGetBalance(accountnum, DefaultBlockParameterName.LATEST)
	  .sendAsync()
	  .get();
	//EthGetBalance balanceWei = web3.ethGetBalance(accountnum,
DefaultBlockParameterName.LATEST).send();
	BigInteger wei = ethGetBalance.getBalance();

	System.out.println("balance is wei "+wei);
	BigDecimal balanceInEther = Convert.fromWei(wei.toString(), Unit.ETHER);
	System.out.println("balance is decimal  "+balanceInEther);

	return String.valueOf(balanceInEther);
}
}
```

6.7 Assign Balance TO User

```
package blockchainevoting;

import java.math.BigDecimal;
import java.sql.ResultSet;
import java.util.ArrayList;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.web3j.crypto.Credentials;
import org.web3j.crypto.Sign;
import org.web3j.crypto.WalletUtils;
import org.web3j.protocol.Web3j;
import org.web3j.protocol.core.methods.response.TransactionReceipt;
import org.web3j.protocol.http.HttpService;
import org.web3j.tx.Transfer;
import org.web3j.utils.Convert;

import Database.DatabaseConnection;

public class AssignBalanceToUser {
 private static final Logger log = LoggerFactory.getLogger(Application.class);

public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub


        DatabaseConnection dbcon;
        dbcon=new DatabaseConnection();
    dbcon.dbconnection();
        ArrayList<String> dataofuser=new ArrayList<>();

        String qry="Select * from buyer where id = 3";
        ResultSet rs=dbcon.getResultSet(qry);
        while(rs.next())
        {
                dataofuser.add(rs.getString("accountid"));


        }



         new AssignBalanceToUser().run(dataofuser);
}
 private void run(ArrayList<String> dataofuser) throws Exception {
        // We start by creating a new web3j instance to connect to remote nodes on the network.
            for(String k:dataofuser) {
        Web3j web3j = Web3j.build(new HttpService("http://localhost:8545"));
        log.info("Connected to Ethereum client version: "
                + web3j.web3ClientVersion().send().getWeb3ClientVersion());
        Credentials credentials =
                WalletUtils.loadCredentials(
                        "abcd",
                        config.keypath+"UTC--2019-11-11T09-39-38.582689000Z--
e5b2b2ce5130fca7817da53cb2b859568f0614db");
        log.info("Credentials loaded");
        log.info("Sending Ether ..");

        String privateKey = credentials.getEcKeyPair().getPrivateKey().toString(16);

        web3j.dbPutHex("hi", "hi1", "hi2");
        web3j.dbPutString("str1", "str1", "str1");
        System.out.println(web3j.ethAccounts().send().getAccounts());

        TransactionReceipt transferReceipt = Transfer.sendFunds(
                web3j, credentials,
               k,  // you can put any address here
                BigDecimal.valueOf(300), Convert.Unit.ETHER)  // 1 wei = 10^-18 Ether
                .sendAsync().get();

        String message = "\\x19Ethereum Signed Message:\n";

        byte[] data = message.getBytes();

        Sign.SignatureData signature = Sign.signMessage(data, credentials.getEcKeyPair());

//              TransactionReceipt transferReceipt1 = Transfer.sendFunds(
//                      web3j, credentials,
//                      "0x57a751ff0a996ea3560a4869c984e25f3c6217c3",  // you can put any address here
//                      BigDecimal.valueOf(3), Convert.Unit.FINNEY)  // 1 wei = 10^-18 Ether
//                      .sendAsync().get();

        System.out.println( transferReceipt.getTransactionHash());
        log.info("Transaction complete : "
                + transferReceipt.getTransactionHash());
    }
 }
}
```

6.8 Payment Code

```
package buyer;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.concurrent.ExecutionException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;



import com.mysql.jdbc.Connection;
import com.mysql.jdbc.PreparedStatement;
import com.mysql.jdbc.Statement;

import Database.DatabaseConnection;
import blockchainevoting.PuttingVoteinBlockchain;
import blockchainevoting.gettingBalance;

/**
 * Servlet implementation class Payment
 */
@WebServlet("/Payment")
public class Payment extends HttpServlet {
private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Payment() {
        super();
        // TODO Auto-generated constructor stub
    }

/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
}

/**
```

```
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub

        PrintWriter out = response.getWriter();
        HttpSession session=request.getSession();
        int auto_id=0;
        String bid = request.getParameter("bid");
        String pid = request.getParameter("pid");
        String billingname = request.getParameter("nameforbill");
        String billaddress = request.getParameter("address");
        String streetname = request.getParameter("streetname");
        String pin = request.getParameter("pin");
        String add_bill = billaddress + " "+streetname+"-"+pin;

        String totalbill = request.getParameter("finalpayment");
        String nameoncard = request.getParameter("nameoncard");
        String cardno = request.getParameter("cardno");
        String expDate = request.getParameter("expDate");
        String cvv = request.getParameter("cvv");

        DatabaseConnection db = new DatabaseConnection();
        Connection con = (Connection) db.dbconnection();
        PreparedStatement stmt;


        String accountinfo=(String)session.getAttribute("accountid");
        System.out.println(accountinfo);
        double totalbilldouble=Double.parseDouble(totalbill);
        totalbilldouble = totalbilldouble/100;
        System.out.println("tot "+totalbilldouble);
        String balance="";
        try {
                 balance=gettingBalance.getbalanceofuser(accountinfo);
        } catch (InterruptedException | ExecutionException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
        }
        double balancedouble=Double.parseDouble(balance);
        System.out.println("bal "+balancedouble);
        if(balancedouble>totalbilldouble) {

        String sellerinfo=(String)session.getAttribute("selleraccountid");

        String qry="Select * from admininfo where id="+sellerinfo+"";

        ResultSet rs12= db.getResultSet(qry);
String accountidofseller="",password="",keystorepath="";
                try {
```

```
                if(rs12.next())
                {
                    accountidofseller=rs12.getString("accountid");
                    password=rs12.getString("encpassword");
                    keystorepath=rs12.getString("keystorepath");
                }


                PuttingVoteinBlockchain.dotransaction(accountinfo, accountidofseller, password,
keystorepath, totalbilldouble);

            } catch (SQLException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }




    String bill_query = "insert into billinfo(billingname,address,pid,bid)
values('"+billingname+"','"+add_bill+"',"+pid+","+bid+")";
    try {
        stmt = (PreparedStatement) con.prepareStatement(bill_query,
Statement.RETURN_GENERATED_KEYS);
        stmt.executeUpdate();
        ResultSet rs = stmt.getGeneratedKeys();
        if(rs.next()) {
          auto_id = rs.getInt(1);
          System.out.println(auto_id);
        }

    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    int pid1=0,price=0, i=0;
    String query1= "select * from cartdetails where buyerid ="+bid;
    ResultSet rs1= db.getResultSet(query1);
    try {
        while(rs1.next())
        {
            pid1 = rs1.getInt("pid");
            price = Integer.parseInt(rs1.getString("price"));
```

```
                        String payment_query = "insert into paymentdetails(amount,cardholdername,cardno,expirydate,cvv,billaddressid,bid,pid) values('"+price+"','"+nameoncard+"','"+cardno+"','"+expDate+"','"+cvv+"',"+auto_id+","+bid+","+pid1+")";
                        i += db.getUpdate(payment_query);

                }
        } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }



        if(i>0)
        {
                String del_query = "delete from cartdetails where buyerid ="+bid;
                db.getUpdate(del_query);

                out.println("<script type=\"text/javascript\">");
                out.println("alert('Order Placed Successfully')");
                out.println("location=\"Buyer/mainPage.jsp\"");
                out.println("</script>");
        }
}
        else
        {

                out.println("<script type=\"text/javascript\">");
                out.println("alert('Balance in blockchain is insufficient')");
                out.println("location=\"Buyer/mainPage.jsp\"");
                out.println("</script>");

        }
}

}
```

6.9 Cart

```
<%@page import="java.util.ArrayList"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<%@ include file="header1.jsp"%>
<%@ page import="Database.DatabaseConnection" %>
<%@ page import="java.sql.ResultSet"%>

<div class="cart">
<div class="container">
```

```
          <div class="cart-top">
                    <a href="mainPage.jsp"><< home</a>
          </div>

          <div class="col-md-9 cart-items">
                    <h2>My Shopping Bag </h2>

                              <script>$(document).ready(function(c) {
                                        $('.close1').on('click', function(c){
                                                  $(this).fadeOut('slow', function(c){
                                                            $(this).remove();
                                                  });
                                                  });
                                        });
                    </script>
<%

String brand="", imagename="",  price="", fueltype="";
int id=0;
ArrayList pidl_ist = new ArrayList();
DatabaseConnection db = new DatabaseConnection();
db.dbconnection();
int bid =(Integer) session.getAttribute("bid");
String pid = request.getParameter("productid");
String query =  "SELECT * FROM cartdetails WHERE buyerid = "+bid;
//System.out.println(query);
ResultSet rs = db.getResultSet(query);
while(rs.next())
{
          id =Integer.parseInt(rs.getString("id"));
          brand = rs.getString("brand");
          price = rs.getString("price");
          imagename = rs.getString("image");
          //fueltype = rs.getString("fueltype");

%>
                    <div class="cart-header" >
                              <div class="close1"></div>
                              <div class="cart-sec">
                                                  <div class="cart-item cyc">
                                                            <img src="image123/<%=imagename%>"/>
                                                  </div>
                                        <div class="cart-item-info">
                                                            <h3><%=brand%><span><%=brand%></span></h3>
                                                            <h4><span>Rs. </span><%=price%></h4>
                                                            <a          href="removeItem.jsp?id=<%=id          %>"><p
class="qty"></p>
                                                            <input   type="submit"   class="form-control   input-small"
value="Remove"></a>
                                        </div>
                                        <div class="clearfix"></div>
                                        <div class="delivery">
                                                <p>Service Charges:: Rs.150.00</p>

                                                <div class="clearfix"></div>
                        </div>
                        </div>
                </div>
<%      }%>
        </div>

        <div class="col-md-3 cart-total">
                <a class="continue" href="mainPage.jsp">Continue to basket</a>
                <div class="price-details">
                        <h3>Price Details</h3>
                        <%
                                int total=0;
                                String q1 = "select sum(price) as t1 from cartdetails where buyerid =
"+bid;
                                ResultSet rs1 = db.getResultSet(q1);
                                if(rs1.next())
                                {
                                        total = rs1.getInt("t1");
                                }
                         %>
                        <span>Total</span>
                        <span class="total"><%=total %></span>
                        <span>Discount</span>
                        <span class="total">---</span>
                        <span>Delivery Charges</span>
                        <span class="total">150</span>
                        <div class="clearfix"></div>
                </div>
                <h4 class="last-price">TOTAL</h4>
                <span class="total final"><%=total=total+150 %></span>
                <div class="clearfix"></div>
                <a                                                                  class="order"
href="payment.jsp?totalbill=<%=total%>&bid=<%=bid%>&pid=<%=pid%>">Place Order</a>
                <!-- <div class="total-item">
                        <h3>OPTIONS</h3>
                        <h4>COUPONS</h4>
                        <a class="cpns" href="#">Apply Coupons</a>
                        <p><a href="#">Log In</a> to use accounts - linked coupons</p>
                </div> -->
                </div>
</div>
</div>
<!---->

<%@ include file="footer.jsp"%>
```

6.10 Remove Item

```
package buyer;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.concurrent.ExecutionException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;



import com.mysql.jdbc.Connection;
import com.mysql.jdbc.PreparedStatement;
import com.mysql.jdbc.Statement;

import Database.DatabaseConnection;
import blockchainevoting.PuttingVoteinBlockchain;
import blockchainevoting.gettingBalance;

/**
 * Servlet implementation class Payment
 */
@WebServlet("/Payment")
public class Payment extends HttpServlet {
private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Payment() {
        super();
        // TODO Auto-generated constructor stub
    }

/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
}

/**
```

```
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    // TODO Auto-generated method stub

    PrintWriter out = response.getWriter();
    HttpSession session=request.getSession();
    int auto_id=0;
    String bid = request.getParameter("bid");
    String pid = request.getParameter("pid");
    String billingname = request.getParameter("nameforbill");
    String billaddress = request.getParameter("address");
    String streetname = request.getParameter("streetname");
    String pin = request.getParameter("pin");
    String add_bill = billaddress + " "+streetname+"-"+pin;

    String totalbill = request.getParameter("finalpayment");
    String nameoncard = request.getParameter("nameoncard");
    String cardno = request.getParameter("cardno");
    String expDate = request.getParameter("expDate");
    String cvv = request.getParameter("cvv");

    DatabaseConnection db = new DatabaseConnection();
    Connection con = (Connection) db.dbconnection();
    PreparedStatement stmt;


    String accountinfo=(String)session.getAttribute("accountid");
    System.out.println(accountinfo);
    double totalbilldouble=Double.parseDouble(totalbill);
    totalbilldouble = totalbilldouble/100;
    System.out.println("tot "+totalbilldouble);
    String balance="";
    try {
         balance=gettingBalance.getbalanceofuser(accountinfo);
    } catch (InterruptedException | ExecutionException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    double balancedouble=Double.parseDouble(balance);
    System.out.println("bal "+balancedouble);
    if(balancedouble>totalbilldouble) {

    String sellerinfo=(String)session.getAttribute("selleraccountid");

    String qry="Select * from admininfo where id="+sellerinfo+"";

    ResultSet rs12= db.getResultSet(qry);
String accountidofseller="",password="",keystorepath="";
        try {
            if(rs12.next())
            {
                accountidofseller=rs12.getString("accountid");
                password=rs12.getString("encpassword");
                keystorepath=rs12.getString("keystorepath");
            }


            PuttingVoteinBlockchain.dotransaction(accountinfo, accountidofseller, password,
keystorepath, totalbilldouble);

        } catch (SQLException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }




    String bill_query = "insert into billinfo(billingname,address,pid,bid)
values('"+billingname+"','"+add_bill+"',"+pid+","+bid+")";
    try {
        stmt = (PreparedStatement) con.prepareStatement(bill_query,
Statement.RETURN_GENERATED_KEYS);
        stmt.executeUpdate();
        ResultSet rs = stmt.getGeneratedKeys();
        if(rs.next()) {
          auto_id = rs.getInt(1);
          System.out.println(auto_id);
        }

    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    int pid1=0,price=0, i=0;
    String query1= "select * from cartdetails where buyerid ="+bid;
    ResultSet rs1= db.getResultSet(query1);
    try {
        while(rs1.next())
        {
            pid1 = rs1.getInt("pid");
            price = Integer.parseInt(rs1.getString("price"));
```

```
                                String payment_query = "insert into paymentdetails(amount,cardholdername,cardno,expirydate,cvv,billaddressid,bid,pid) values('"+price+"','"+nameoncard+"','"+cardno+"','"+expDate+"','"+cvv+"',"+auto_id+","+bid+","+pid1+")";
                                i += db.getUpdate(payment_query);

                        }
                } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }



                if(i>0)
                {
                        String del_query = "delete from cartdetails where buyerid ="+bid;
                        db.getUpdate(del_query);

                        out.println("<script type=\"text/javascript\">");
                        out.println("alert('Order Placed Successfully')");
                        out.println("location=\"Buyer/mainPage.jsp\"");
                        out.println("</script>");
                }
        }
                else
                {

                        out.println("<script type=\"text/javascript\">");
                        out.println("alert('Balance in blockchain is insufficient')");
                        out.println("location=\"Buyer/mainPage.jsp\"");
                        out.println("</script>");

                }
        }

        }
```

6.11 Logout

```
<%

session.invalidate();
response.sendRedirect("index.jsp");

%>
```

6.12 Databases

```
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;

CREATE DATABASE /*!32312 IF NOT EXISTS*/`smartcontract` /*!40100 DEFAULT CHARACTER SET latin1 */;

USE `smartcontract`;

/*Table structure for table `admininfo` */

DROP TABLE IF EXISTS `admininfo`;

CREATE TABLE `admininfo` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(1000) default NULL,
  `Emailid` varchar(1000) default NULL,
  `phonenumber` varchar(1000) default NULL,
  `password` varchar(1000) default NULL,
  `accountid` longtext,
  `path` longtext,
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*Data for the table `admininfo` */

insert into `admininfo`(`id`,`name`,`Emailid`,`phonenumber`,`password`,`accountid`,`path`) values (1,'Ashish Roy','qwerty19roy@gmail.com','9022055911','123',NULL,NULL);

/*Table structure for table `bikeaccessaries` */

DROP TABLE IF EXISTS `bikeaccessaries`;

CREATE TABLE `bikeaccessaries` (
  `id` int(11) NOT NULL auto_increment,
  `productname` varchar(1000) default NULL,
  `brand` varchar(1000) default NULL,
  `price` varchar(1000) default NULL,
  `image` varchar(1000) default NULL,
  `discription` varchar(1000) default NULL,
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*Data for the table `bikeaccessaries` */

/*Table structure for table `bikedetails` */

DROP TABLE IF EXISTS `bikedetails`;

CREATE TABLE `bikedetails` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(1000) default NULL,
  `price` varchar(1000) default NULL,
  `brand` varchar(1000) default NULL,
  `fueltype` varchar(1000) default NULL,
  `discription` varchar(1000) default NULL,
  `imagename` varchar(1000) default NULL,
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*Data for the table `bikedetails` */

/*Table structure for table `billinfo` */

DROP TABLE IF EXISTS `billinfo`;

CREATE TABLE `billinfo` (
  `billid` int(11) NOT NULL auto_increment,
  `billingname` varchar(100) default NULL,
  `address` varchar(200) default NULL,
  `pid` int(11) default NULL,
  `bid` int(11) default NULL,
  PRIMARY KEY  (`billid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*Data for the table `billinfo` */

insert  into `billinfo`(`billid`,`billingname`,`address`,`pid`,`bid`) values (1,'Manjunath Wagadari','sector
3 airoli vamanrao pai marg-400708',NULL,1),(2,'Manjunath Wagadari','sector 3 airoli vamanrao pai
marg-400708',NULL,1),(3,'Manjunath Wagadari','sector 3 airoli vamanrao pai marg-
400708',NULL,1),(4,'Manjunath Wagadari','mumbai vamanrao pai marg-
400708',NULL,1),(5,'Manjunath Wagadari','mumbai vamanrao pai marg-
400708',NULL,1),(6,'Manjunath Wagadari','mumbai vamanrao pai marg-400708',NULL,1),(7,'ningesh
kharatmol','airoli airoli-400708',NULL,1);

/*Table structure for table `block1` */

DROP TABLE IF EXISTS `block1`;

CREATE TABLE `block1` (
  `id` int(255) NOT NULL auto_increment,
  `user` varchar(1000) default NULL,
  `totalammount` varchar(1000) default NULL,
  `hashcode` varchar(1000) default NULL,
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*Data for the table `block1` */

/*Table structure for table `buyer` */
```

```
DROP TABLE IF EXISTS `buyer`;

CREATE TABLE `buyer` (
  `id` int(10) NOT NULL auto_increment,
  `name` varchar(100) default NULL,
  `email` varchar(100) default NULL,
  `phone` varchar(50) default NULL,
  `password` varchar(100) default NULL,
  `cardNumber` varchar(100) default NULL,
  `address` varchar(500) default NULL,
  `accountid` longtext,
  `path` longtext,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*Data for the table `buyer` */

insert into `buyer`(`id`,`name`,`email`,`phone`,`password`,`cardNumber`,`address`,`accountid`,`path`)
values (1,'abhishek','abhishek@gmail.com','9004850819','123',NULL,'Airoli',NULL,NULL);

/*Table structure for table `caraccessaries` */

DROP TABLE IF EXISTS `caraccessaries`;

CREATE TABLE `caraccessaries` (
  `id` int(11) NOT NULL auto_increment,
  `productname` varchar(1000) default NULL,
  `brand` varchar(1000) default NULL,
  `price` varchar(1000) default NULL,
  `image` varchar(1000) default NULL,
  `discription` varchar(1000) default NULL,
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*Data for the table `caraccessaries` */

insert into `caraccessaries`(`id`,`productname`,`brand`,`price`,`image`,`discription`) values
(1,NULL,NULL,NULL,NULL,NULL);

/*Table structure for table `cardetails` */

DROP TABLE IF EXISTS `cardetails`;

CREATE TABLE `cardetails` (
  `id` int(11) NOT NULL auto_increment,
  `adminid` int(11) default NULL,
  `price` int(11) default NULL,
  `brand` varchar(1000) default NULL,
  `fueltype` varchar(1000) default NULL,
  `discription` varchar(1000) default NULL,
  `imagename` varchar(1000) default NULL,
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*Data for the table `cardetails` */

insert into `cardetails`(`id`,`adminid`,`price`,`brand`,`fueltype`,`discription`,`imagename`) values
(1,1,1000000,'lodgy','petrol','jhdh jadghg asd','lodgy.png'),(2,1,1200000,'safari','cng','fsf fdfs
sf','safari.jpg'),(3,1,1500000,'swift','cng','sdf hgjgrt dfg s dv','swift.gif'),(4,1,1800000,'tuv','diesel','dfeg
fdg sdew','tuv.png'),(5,1,2000000,'zica','petrol','ewrer ef dfs f','zica.jpg');

/*Table structure for table `cartdetails` */

DROP TABLE IF EXISTS `cartdetails`;

CREATE TABLE `cartdetails` (
  `id` int(11) NOT NULL auto_increment,
  `pid` int(11) default NULL,
  `buyerid` int(10) default NULL,
  `brand` varchar(1000) default NULL,
  `price` varchar(1000) default NULL,
  `image` varchar(1000) default NULL,
  `payment` varchar(1000) default NULL,
  `hashtext` varchar(1000) default NULL,
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*Data for the table `cartdetails` */

/*Table structure for table `paymentdetails` */

DROP TABLE IF EXISTS `paymentdetails`;

CREATE TABLE `paymentdetails` (
  `id` int(5) NOT NULL auto_increment,
  `amount` int(11) default NULL,
  `cardholdername` varchar(50) default NULL,
  `cardno` varchar(20) default NULL,
  `expirydate` varchar(10) default NULL,
  `cvv` varchar(5) default NULL,
  `billaddressid` int(11) default NULL,
  `bid` int(5) default NULL,
  `pid` int(5) default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*Data for the table `paymentdetails` */

insert into
`paymentdetails`(`id`,`amount`,`cardholdername`,`cardno`,`expirydate`,`cvv`,`billaddressid`,`bid`,`pid`)
values (4,1500000,'manjunath Wagadari
','45678956321456523','12/23','312',6,1,3),(5,1000000,'manjunath Wagadari
```

```
','45678956321456523','12/23','312',6,1,1),(6,1500000,'abcd','abcd','11/29','123',7,1,3),(7,1000000,'abcd','
abcd','11/29','123',7,1,1);

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
```

Chapter 7

Results

7. Results

This chapter contains results of final product module wise. It contains snapshots of different modules of the system.

7.1 Login and Registration Form

Fig 7.1 Shows the Registration form for the user who wants to use our service. At the time of registration,user have to enter some basic and required details like name, email,phonenumber, and Password.It also has a consent for admin for accessing the data.

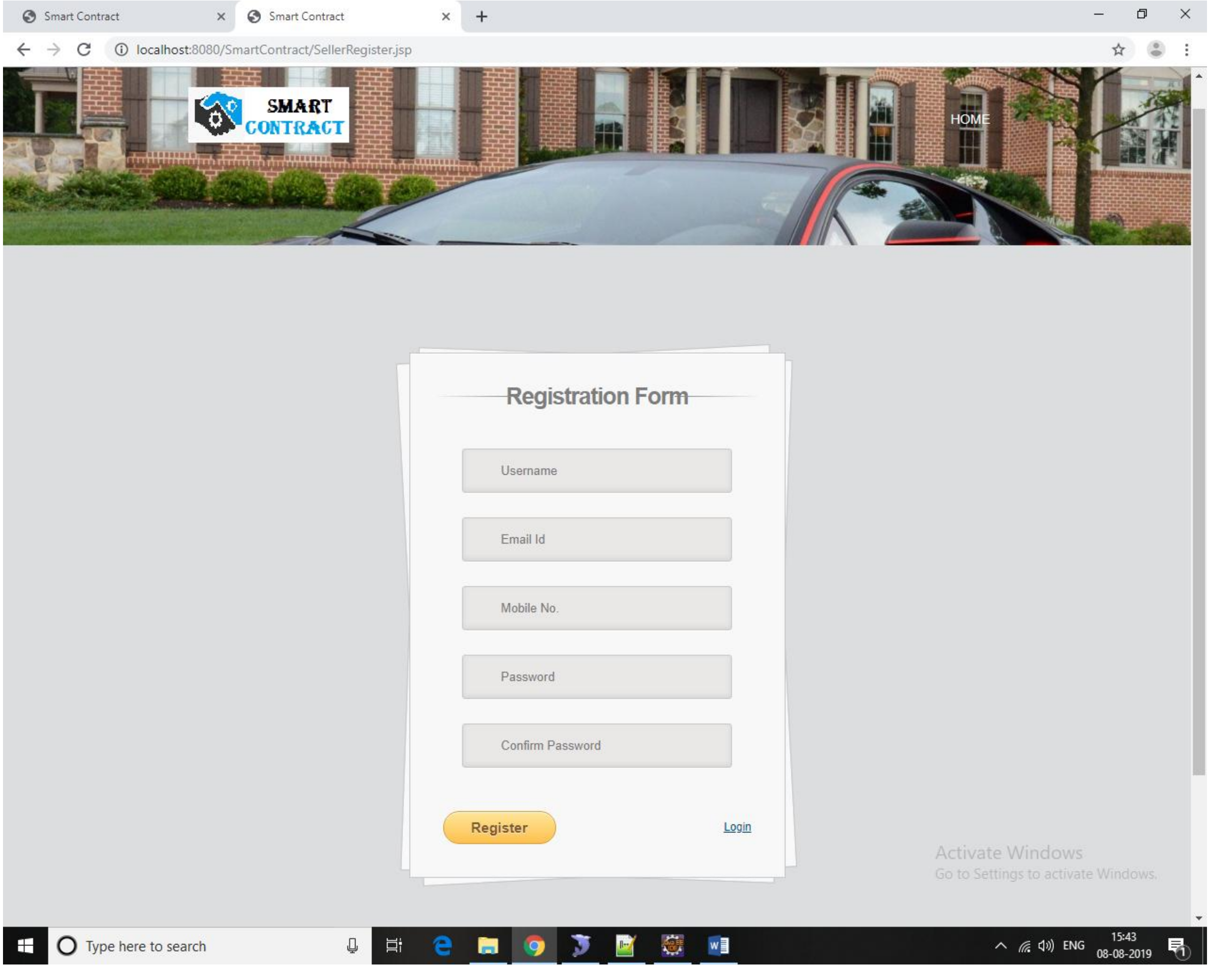


Figure 7.1 Registration form

Fig 7.2 Show the login form for the users that have registered already and for login purpose users have to enter their email address and password.This page also has a link to registration purpose if the user is not registered.

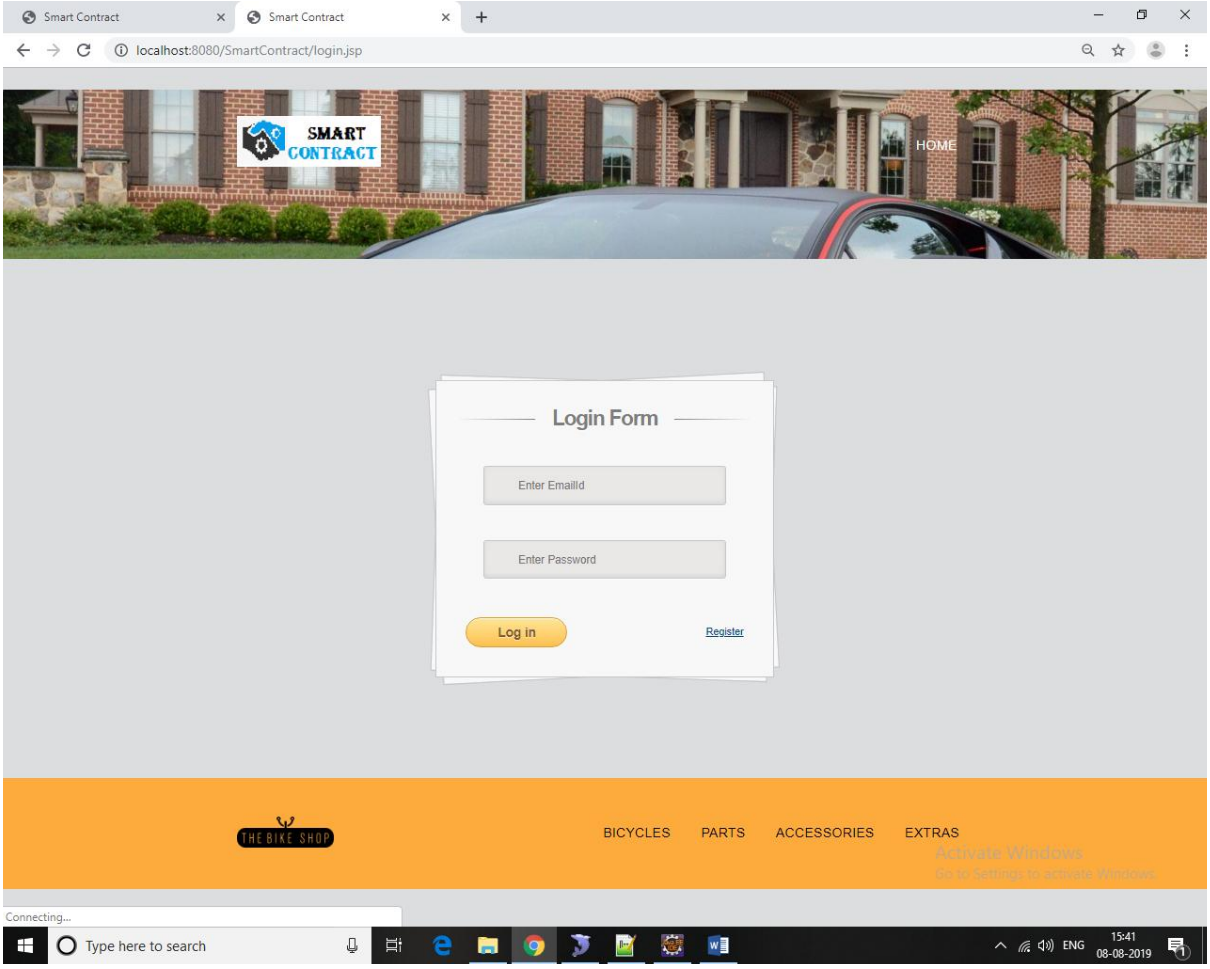


Figure 7.2 Login form

7.2 Dashboard page

After successful login user will be redirect to the dashboard and it contains various products/cars, and it also contains various operations like Home, Accessories, Add new car and Logout functionalities. One can go through different models through the main page and perform their purchase.

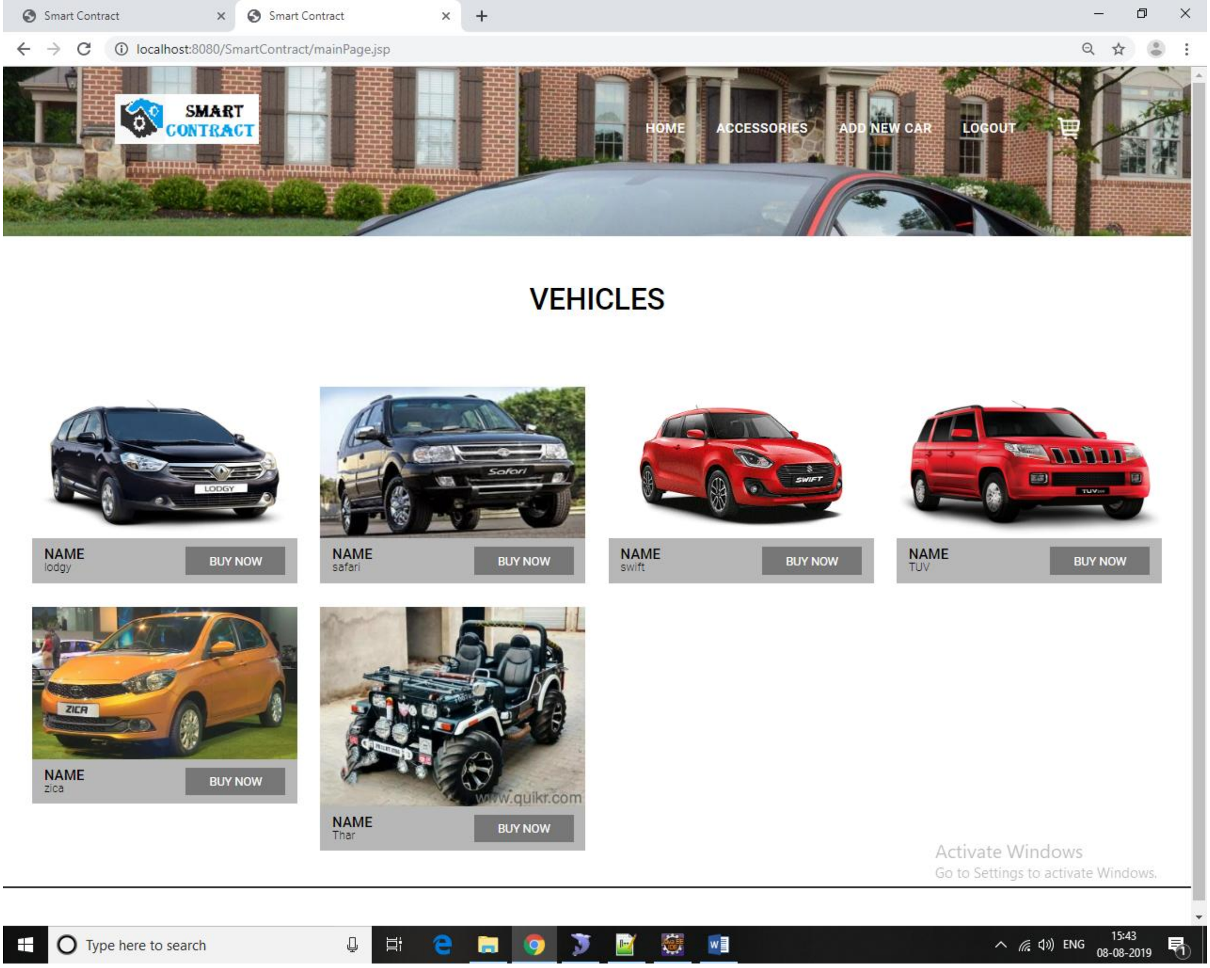
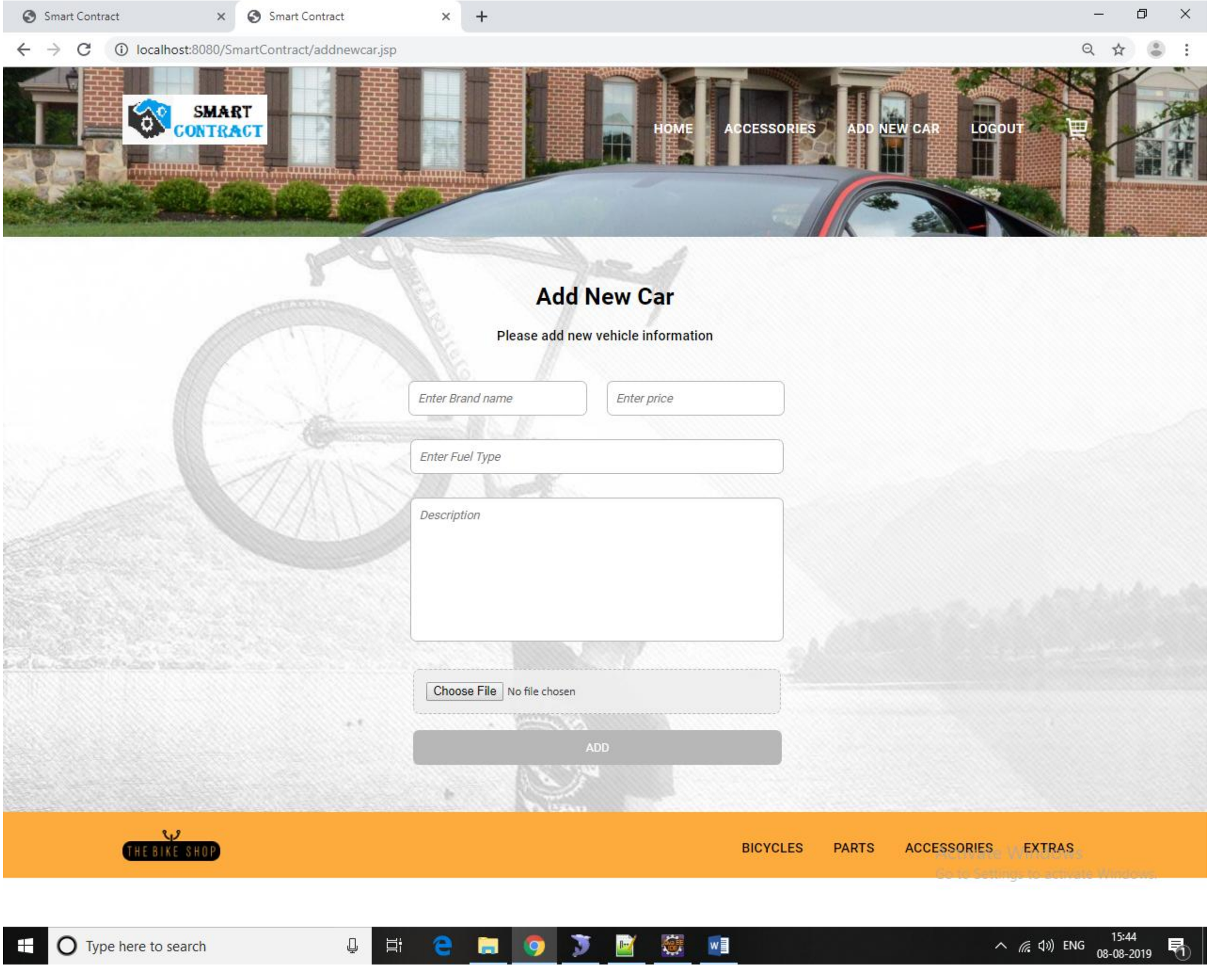


Figure 7.3 Dashboard page

7.3 Create and update user profile

Fig 8.4 shows the functionality of the user page ,in this page user can fill in the details of their product along with the image. They need to fill their own detailsto the website at the time of registration and it displays the users and its information’s likeBrand name, price, fuel type, image etc.



Add New Car

Please add new vehicle information

Enter Brand name

Enter price

Enter Fuel Type

Description

Figure 7.4 Update User Profile

Chapter 9

Conclusion

9. Conclusion

This system can be incorporated with provision access protocols to provide a secure and private car trading environment without the need of any intermediary. We have shown that Bit-Contracts provides all major functionalities that are required for a car trading platform, and provides security and privacy by design. The sum of the entire cost of deploying and using our system on the Ethereum network is Rs. 27.16 which in comparison is relatively cheaper to the commission fee paid to large organizations. Hence, we conclude that along with being functionally sound, secure and private, Bit-Contracts is also cost effective for its users. As future work we would like to advance our system design and implementation to work with fully encrypted booking details, including renting the car with the price per day of the car, price per extra day and required number of days which are used for calculation of payments. Another potential direction could be to adapt Bit-Contracts so that it supports the use of advanced cryptographic primitives such as zero-knowledge proofs.

References

[1] Innovation Hub. (2017, January) Innovation Hub. [Online]. https://maestrolabs.io/blockchain/

[2] Dr. Gavin Wood, Ethereum [Yellow Paper], 2019.

[3] Vitalik Buterin, *Ethereum White Paper - A Next Generation Smart Contract & Decentralized Application Platform*.: Ethereum, 2013.

[4] (2017, August) TechTarget. [Online]. https://searchcio.techtarget.com/definition/distri buted-ledger

[5] CoinMarketCap. [Online]. https://coinmarketcap.com/

[6] Leonid Astakhov. (2018, July) Intrachain. [Online]. https://medium.com/intrachain-insights/how-werevolutionize-accounting-through-smartcontracts-7268c7e87969

[7] Filipe Calvão, "Crypto-miners: Digital labor and the power," *Economic Anthropology*, vol. 6, no. 1, pp. 123-134, 2019.

[8] Iraklis Symeonidis et al., "SePCAR: A Secure and Privacy-Enhancing Protocol for Car Access Provision," in *Computer Security – ESORICS 2017*. Oslo, Norway: Springer, Cham, August 2017, vol. 10493, pp. 475-493.

[9] Rui Zhang, Rui Xue, and Ling Lui, "Security and Privacy on Blockchain," *ACM Computing Surveys*, vol. 1, no. 1, p. 35, January 2019.

[10] (2020) CoinGecko. [Online]. https://www.coingecko.com/en/coins/ethereum/i nr

[11] Ameer Rosic. (2018) Blockgeeks. [Online]. https://blockgeeks.com/guides/ethereum-gas/

[12] Remix-IDE. [Online]. https://remix.ethereum.org/