SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

Bachelor of Technology (Ord) Networking Technologies

Module: COMP3001 – Final Report

YEAR 3

Project Title:

Car Mileage Blockchain System

Date: 08/02/2021

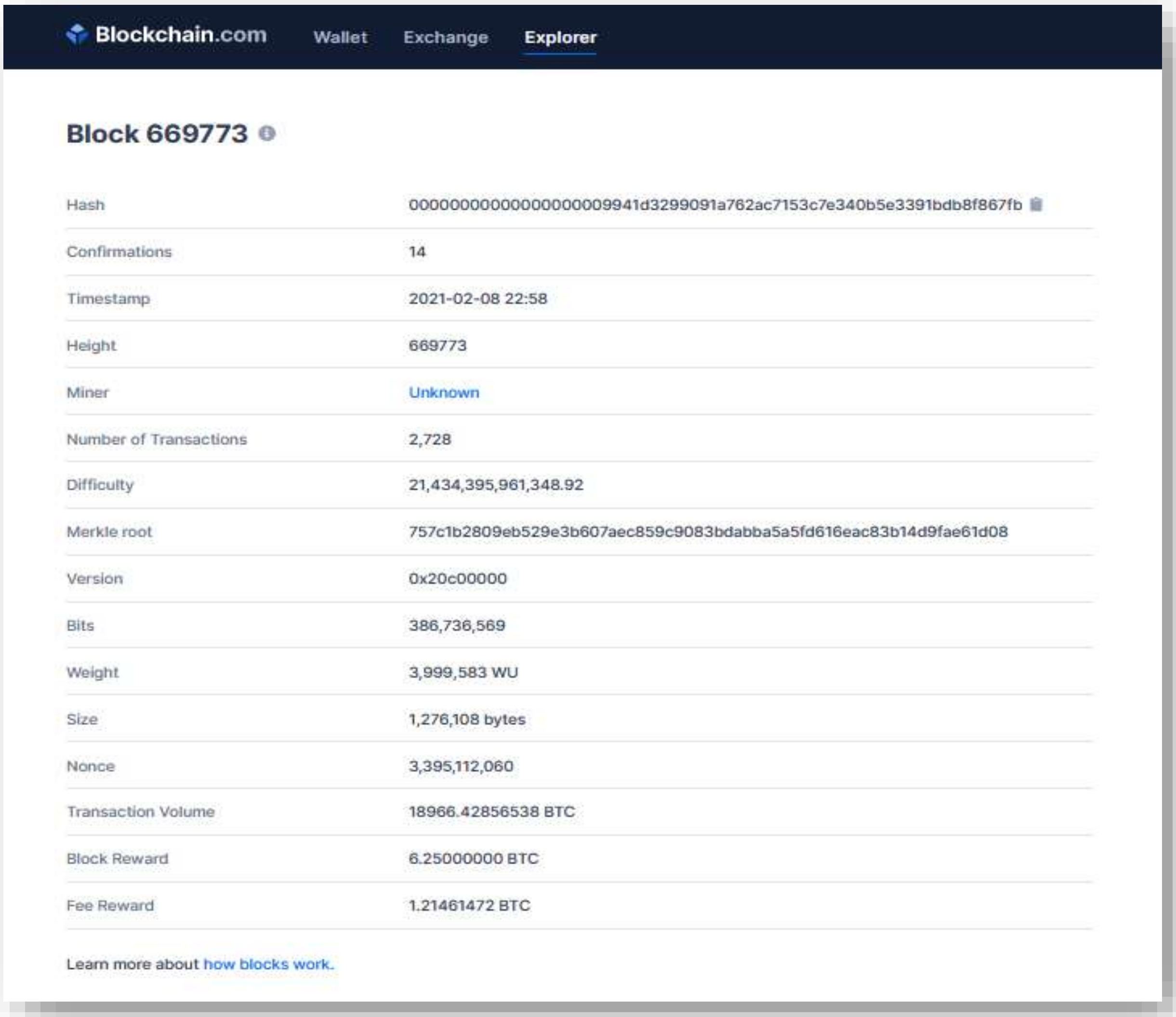
Word Count: 4159

Table of Contents

Abstract

The term blockchain describes a method of securely storing a list of record type data. Individual records can be collated into a “block” which in addition to these records usually contains a timestamp and cryptographic hash made from the data of the previous block in the chain (e.g. SHA256 or BLAKE2B). A blockchain infrastructure can be public, private, or various mixtures of the two (as you will see in our completed project), the overall goal of this infrastructure Is to host a “chain” of sequential records or a ledger of a specific type of information.

Probably the most well-known usage of the blockchain infrastructure today is that relating to virtual economies or “cryptocurrency”, the most prominent of which being Bitcoin (BTC) and Ethereum (ETH) among thousands of others. Here the blockchain works as a publicly accessible ledger containing the entire transactional history of the cryptocurrency. A block contains high level info contained within:



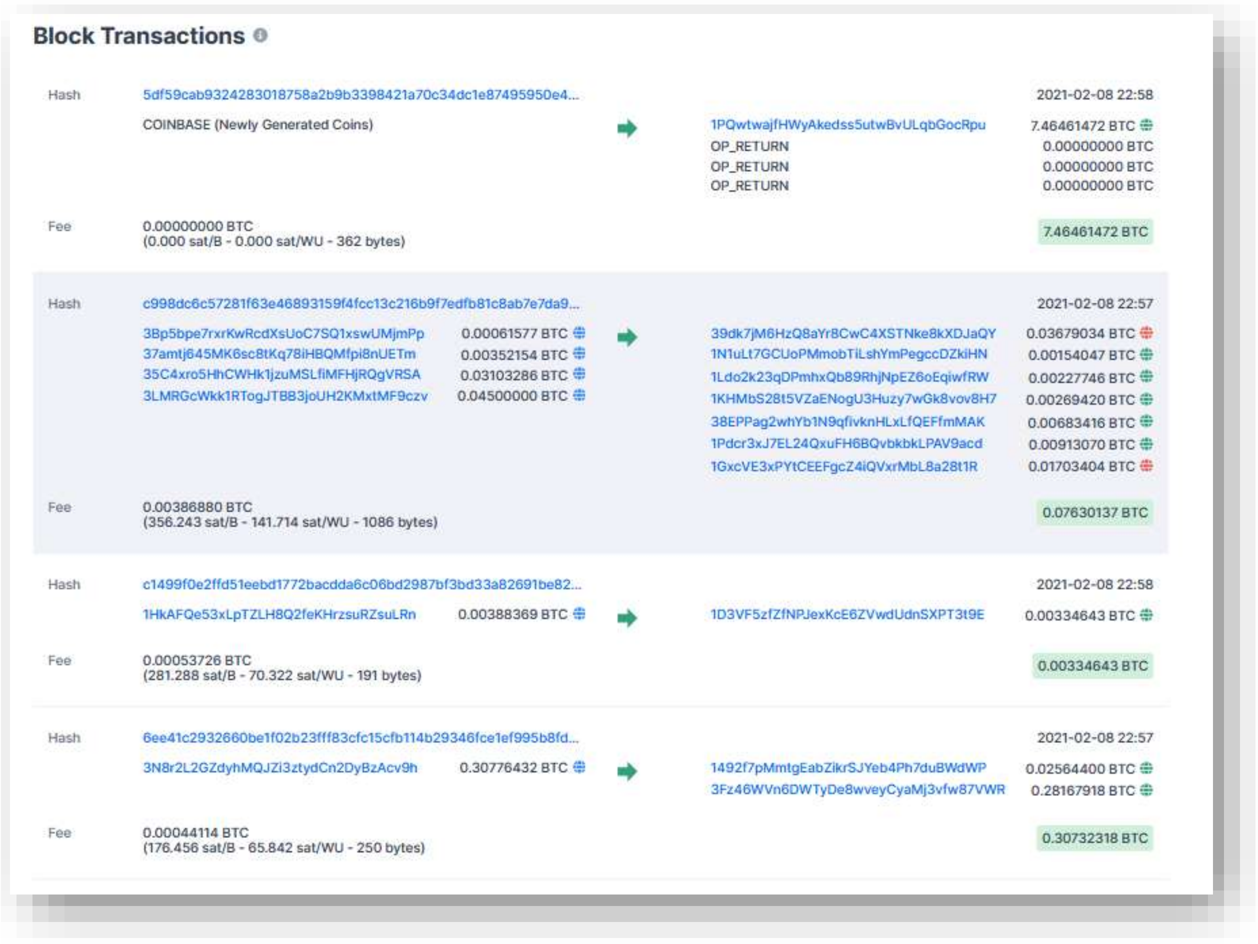
Block 669773

Hash	0000000000000000000099941d3299091a762ac7153c7e340b5e3391bdb8f867fb
Confirmations	14
Timestamp	2021-02-08 22:58
Height	669773
Miner	Unknown
Number of Transactions	2,728
Difficulty	21,434,395,961,348.92
Merkle root	757c1b2809eb529e3b607aec859c9083bdabba5a5fd616eac83b14d9fae61d08
Version	0x20c00000
Bits	386,736,569
Weight	3,999,583 WU
Size	1,276,108 bytes
Nonce	3,395,112,060
Transaction Volume	18966.42856538 BTC
Block Reward	6.25000000 BTC
Fee Reward	1.21461472 BTC

Learn more about how blocks work.

Figure 1: Block Structure

Furthermore a block is further divided into its “records” which in the case of BTC refers to the transactions:



Block Transactions

```
Hash   5df59cab9324283018758a2b9b3398421a70c34dc1e87495950e4...                                          2021-02-08 22:58
       COINBASE (Newly Generated Coins)                          →   1PQwtwajfHWyAkedss5utwBvULqbGocRpu   7.46461472 BTC
                                                                     OP_RETURN                            0.00000000 BTC
                                                                     OP_RETURN                            0.00000000 BTC
                                                                     OP_RETURN                            0.00000000 BTC
Fee    0.00000000 BTC
       (0.000 sat/B - 0.000 sat/WU - 362 bytes)                                                           7.46461472 BTC

Hash   c998dc6c57281f63e46893159f4fcc13c216b9f7edfb81c8ab7e7da9...                                        2021-02-08 22:57
       3Bp5bpe7rxrKwRcdXsUoC7SQ1xswUMjmPp  0.00061577 BTC    →   39dk7jM6HzQ8aYr8CwC4XSTNke8kXDJaQY   0.03679034 BTC
       37amtj645MK6sc8tKq78iHBQMfpi8nUETm  0.00352154 BTC        1N1uLt7GCUoPMmobTiLshYmPegccDZkiHN   0.00154047 BTC
       35C4xro5HhCWHk1jzuMSLfiMFHjRQgVRSA  0.03103286 BTC        1Ldo2k23qDPmhxQb89RhjNpEZ6oEqiwfRW   0.00227746 BTC
       3LMRGcWkk1RTogJTBB3joUH2KMxtMF9czv  0.04500000 BTC        1KHMbS28t5VZaENogU3Huzy7wGk8vov8H7   0.00269420 BTC
                                                                 38EPPag2whYb1N9qfivknHLxLfQEFfmMAK   0.00683416 BTC
                                                                 1Pdcr3xJ7EL24QxuFH6BQvbkbkLPAV9acd   0.00913070 BTC
                                                                 1GxcVE3xPYtCEEFgcZ4iQVxrMbL8a28t1R   0.01703404 BTC
Fee    0.00386880 BTC
       (356.243 sat/B - 141.714 sat/WU - 1086 bytes)                                                      0.07630137 BTC

Hash   c1499f0e2ffd51eebd1772bacdda6c06bd2987bf3bd33a82691be82...                                         2021-02-08 22:58
       1HkAFQe53xLpTZLH8Q2feKHrzsuRZsuLRn  0.00388369 BTC    →   1D3VF5zfZfNPJexKcE6ZVwdUdnSXPT3t9E   0.00334643 BTC
Fee    0.00053726 BTC
       (281.288 sat/B - 70.322 sat/WU - 191 bytes)                                                        0.00334643 BTC

Hash   6ee41c2932660be1f02b23fff83cfc15cfb114b29346fce1ef995b8fd...                                       2021-02-08 22:57
       3N8r2L2GZdyhMQJZi3ztydCn2DyBzAcv9h  0.30776432 BTC    →   1492f7pMmtgEabZikrSJYeb4Ph7duBWdWP   0.02564400 BTC
                                                                 3Fz46WVn6DWTyDe8wveyCyaMj3vfw87VWR   0.28167918 BTC
Fee    0.00044114 BTC
       (176.456 sat/B - 65.842 sat/WU - 250 bytes)                                                        0.30732318 BTC
```

Figure 2: Individual Records

*Glossary of BTC terms can be found here: https://support.blockchain.com/hc/en-us/articles/213276463-Bitcoin-Glossary

Background of Project

The usage of a Blockchain type infrastructure isn't restricted to just cryptocurrency operation, it has found many other uses in fields such as:

- Supply Chain Management e.g. IBM Food Trust: https://www.ibm.com/blockchain/solutions/food-trust

- Decentralized Digital Identity e.g. Hyperledger Indy: https://www.hyperledger.org/use/hyperledger-indy

- Decentralized Healthcare Records e.g. MediBloc: https://coincentral.com/medibloc-beginner-guide/

- Electronic Voting: https://www.ledgerinsights.com/japans-tsukuba-city-to-use-blockchain-based-electronic-voting/

- Notary Functions for many businesses such as Copywriters, Insurance firms, Music artists e.g. stampd.io: https://stampd.io/ and SilentNotary: https://www.silentnotary.com/

In all applicable areas the blockchain plays a specific role, that of highly secure, highly reliable information store, hence the reason for its adoption into many different areas. Traditionally this information is stored in a decentralized fashion meaning no single entity has control over the platform as a whole, this is usually accomplished by the use of a *peer-to-peer* (P2P) framework where any client can setup and run their own "*node*" that participates in the blockchain network where each node is assigned a unique ID for identification and authentication purposes.

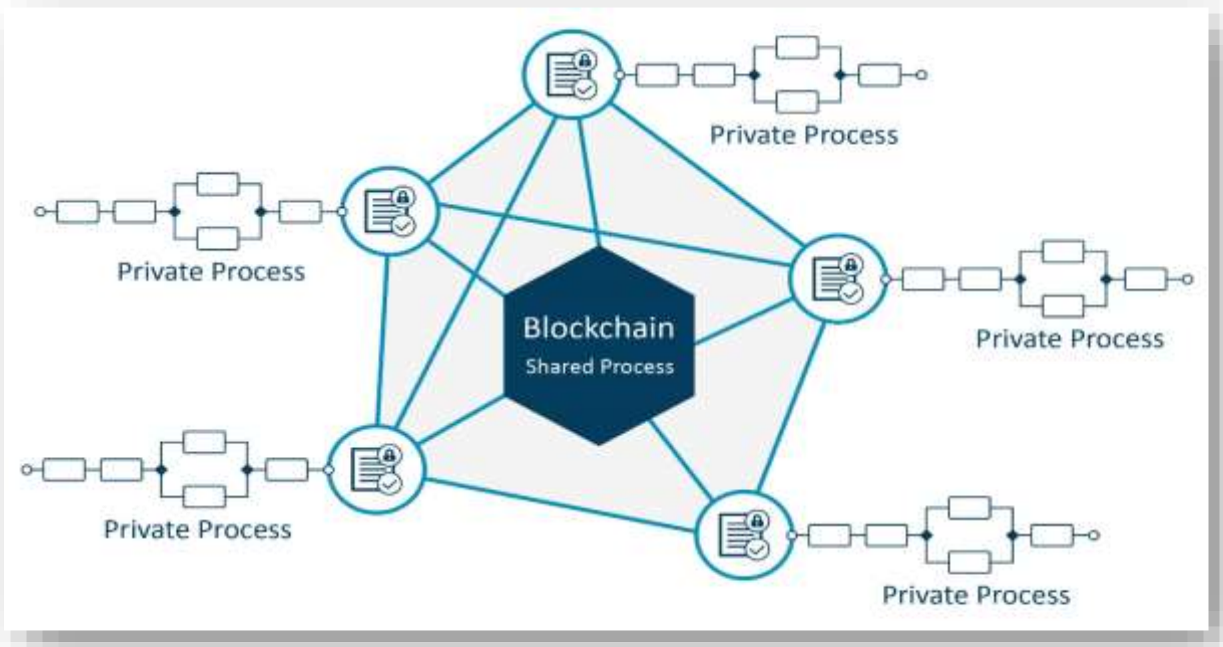


Figure 3: Example of Blockchain network

In order for this type of operation to function each node/client that is participating needs to operate on the same copy of the blockchain as every other node, in other words, an *authoritative* blockchain needs to be established among each participant. Usually the longest blockchain among all nodes is established as the authoritative, this is in part due to the way in which new blocks are created in the blockchain, a process referred to as “*mining*”. By each node working on their own copy of the authoritative blockchain it reduces the chance of a single entity gaining control over the blockchain framework:

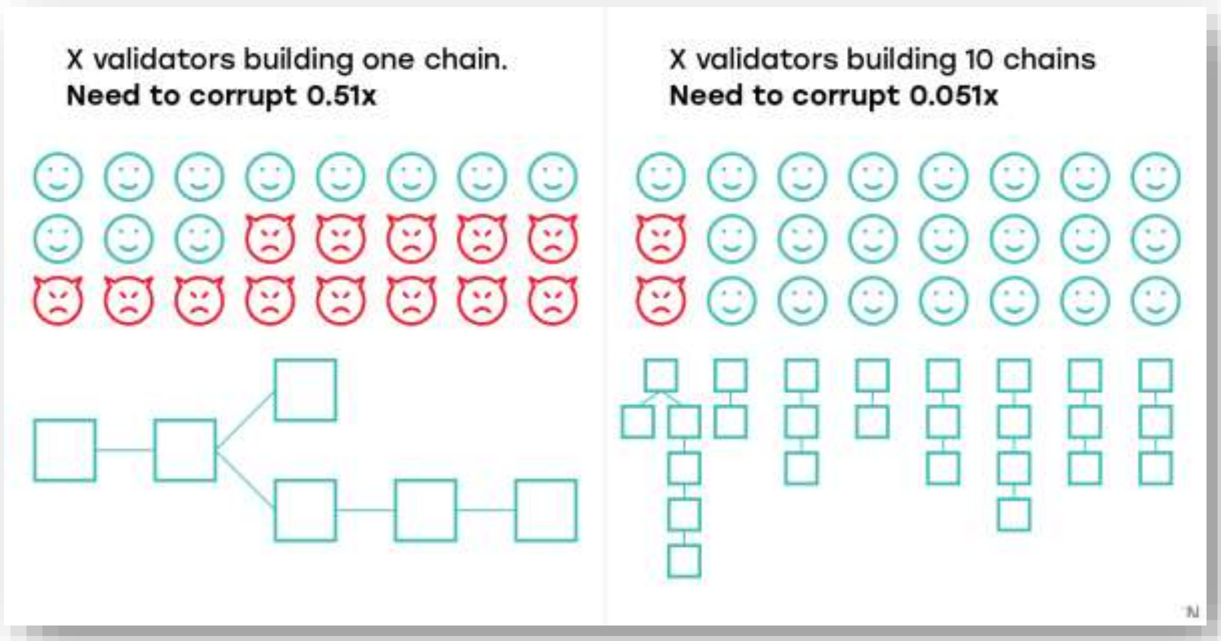

X validators building one chain.
Need to corrupt 0.51x
X validators building 10 chains
Need to corrupt 0.051x

Figure 4: Single vs. Multiple copies of Blockchain

Technically any node participating in the blockchain network can create a new block, however if new blocks could be created instantly there would be little to prevent a single node from creating as many blocks as needed to overwrite and establish their own authoritative blockchain within the network.

To counteract this, blockchains make use of a single or multiple consensus algorithms – a certain protocol followed by each node within the blockchain to establish trust and consistency. A typical consensus algorithm is *Proof of Work* (PoW), in PoW a *block creator* (miner) has to solve a specific computational problem to find a certain value (Proof) that conveys to other participants that a degree of work has been performed. This increases time taken to create new blocks as first a problem has to be solved and then validated by other nodes in the P2P network, where the time taken increases with the complexity of the problem to solve.

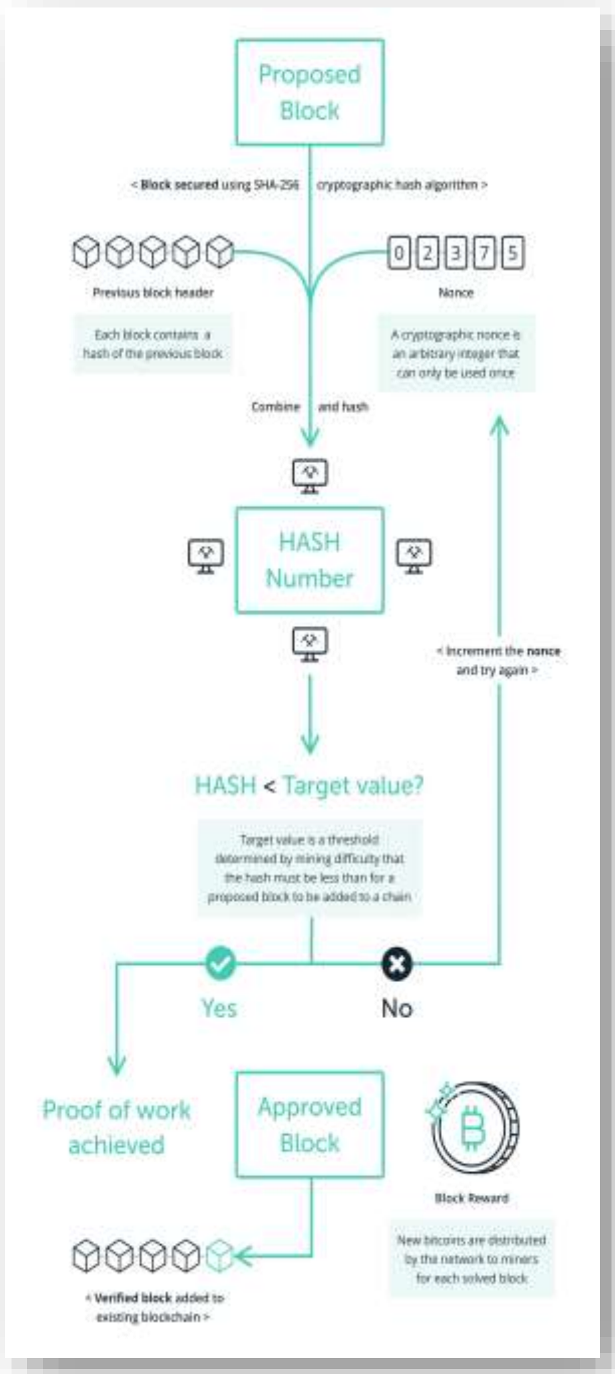

Proposed Block
< Block secured using SHA-256 cryptographic hash algorithm >
0 2 3 7 5
Previous block header
Nonce
Each block contains a hash of the previous block
A cryptographic nonce is an arbitrary integer that can only be used once
Combine and hash
HASH Number
< Increment the nonce and try again >
HASH < Target value?
Target value is a threshold determined by mining difficulty that the hash must be less than for a proposed block to be added to a chain
Yes
No
Proof of work achieved
Approved Block
Block Reward
New bitcoins are distributed by the network to miners for each solved block
< Verified block added to existing blockchain >

Figure 5: PoW Example

**Note: There are various methods for establishing and adding to an authoritative blockchain other than the implementation of a PoW algorithm such as: Proof of Authority (PoA), Proof of Stake (PoS), Smart Contracts etc... however since I went with PoW for this project I've only explained it in this context.*

Each new block in a blockchain contains the hash of the previous block, *hashing* is the process of taking some data and creating an encoded outcome based on it, this outcome Is easily verified (if you put in the same data you will get the same output) but virtually impossible to reverse (to get the original input from the hashed value alone).

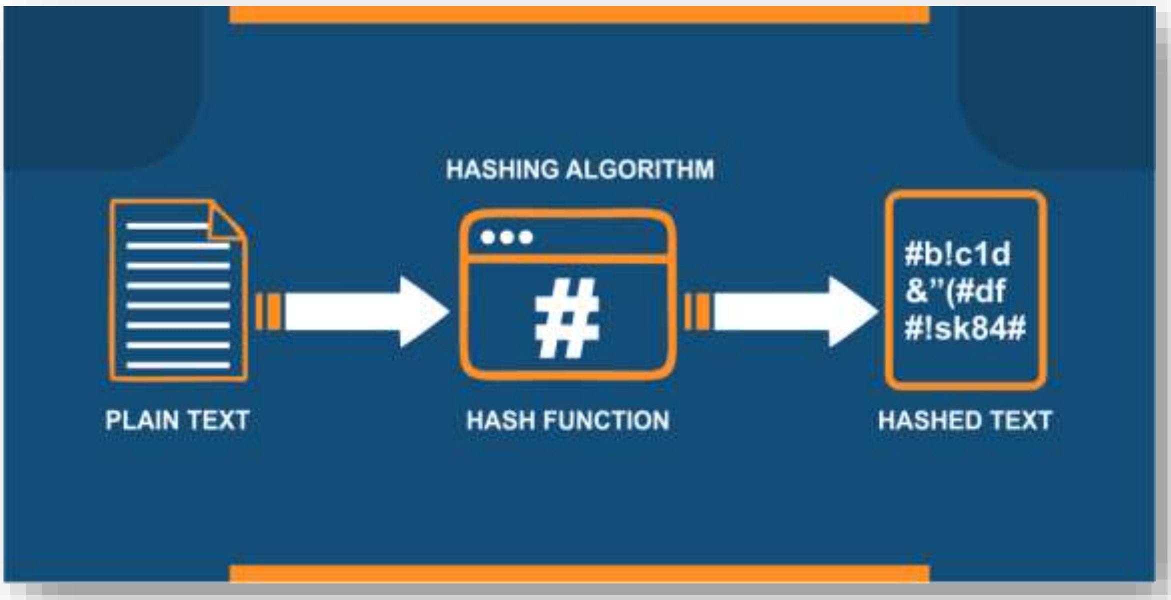


Figure 6: Hashing Process

Before the Blockchain can be used a *genesis* block must be implicitly created, this block serves as a starting point or *seed* for the rest of the chain and contains at minimum a set previous hash and proof value (Can be set to anything).

Proposed Solution

Our proposed project is a Vehicle Mileage Notary system built on a simple blockchain framework delivered over a web-based interface. The blockchain & Webapp functionality will be implemented through a single program using Python's Flask Webapp & Requests HTTP libraries. The solution will incorporate a website interface for interacting with the blockchain and adding information to it. The specific information contained within the records inside the blockchain are: Car Model Number, Chassis Number, & Mileage in Km.

Within this site the Blockchain itself and relevant information should be publicly visible to any entity as the idea behind this project is to provide a secure, authentic information platform for verified Vehicle Mileage Information, however, the functionality for adding records and creating new blocks will be reserved for authorized entities such as Vehicle Information Authorities. The basic differences of these privileges are outlined below:

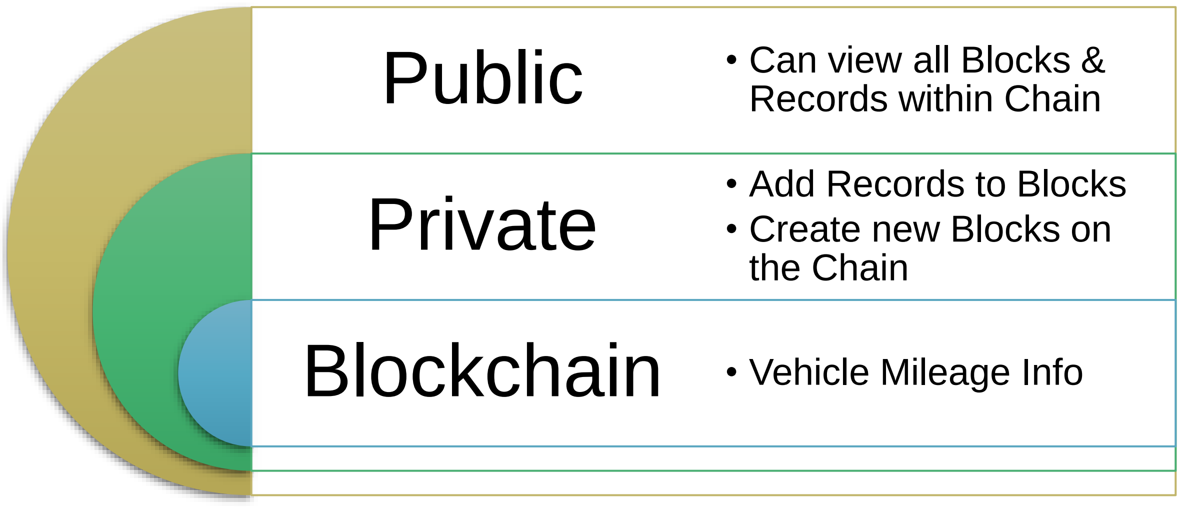

Public
• Can view all Blocks & Records within Chain
Private
• Add Records to Blocks
• Create new Blocks on the Chain
Blockchain
• Vehicle Mileage Info

Figure 7: Blockchain Access Privileges

While the main structure of the program/platform can be thought of as:

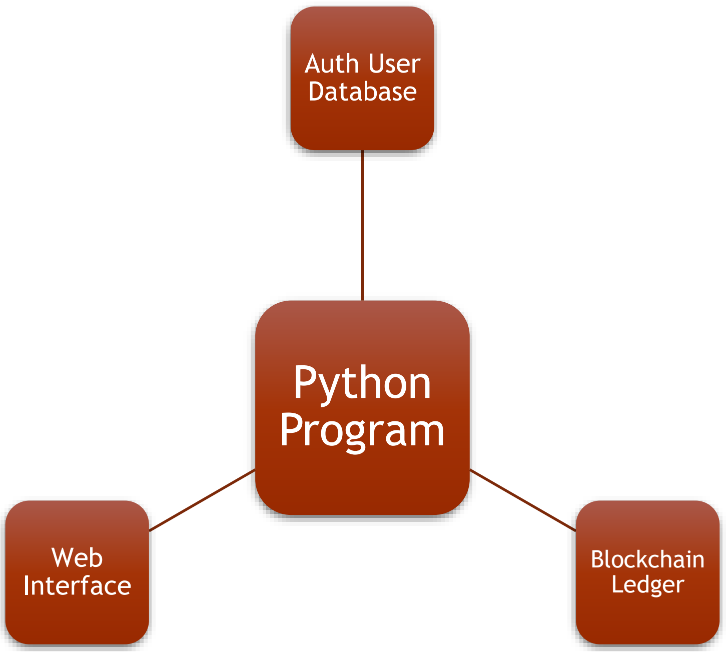


Figure 8: Program/Platform Structure

The operation of the blockchain, webapp, and users will resemble:

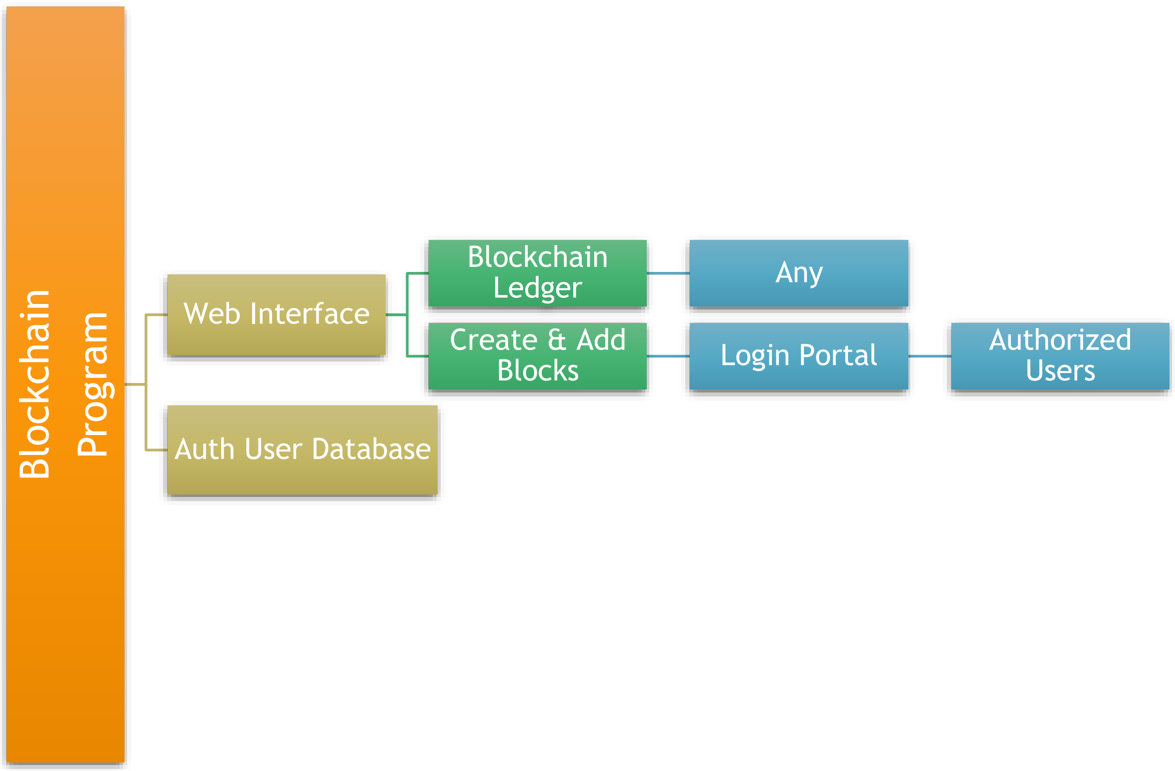


Figure 9: Operational Structure

The Implementation

Python Dependencies:
Flask: https://flask.palletsprojects.com/en/1.1.x/
Requests: https://2.python-requests.org/en/master/

Python Blockchain Class

In our main program, called **blockchain_flask_app.py**, starting off we need to develop a Blockchain class, this will be done in Python and will need the following functionality:

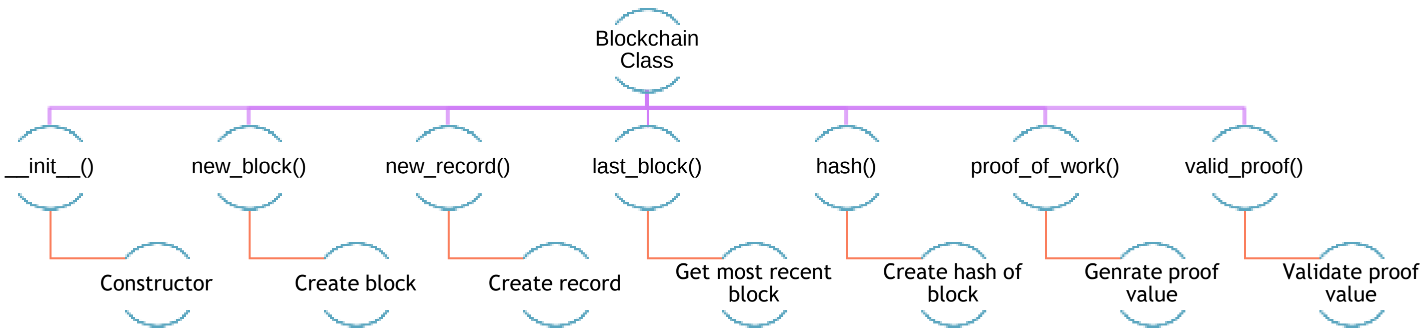
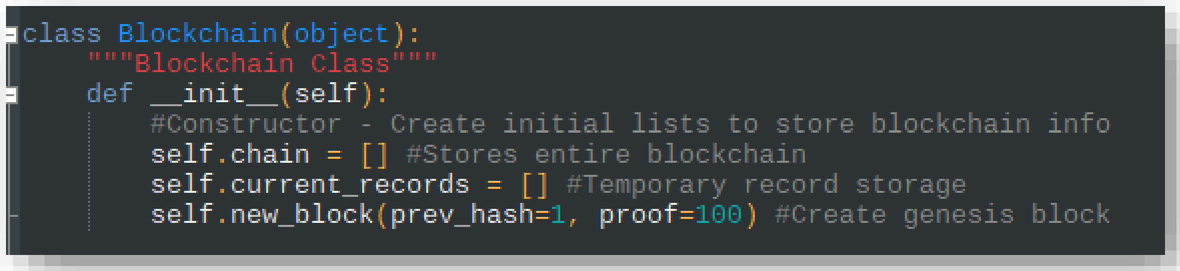


Figure 10: Blockchain Class Structure

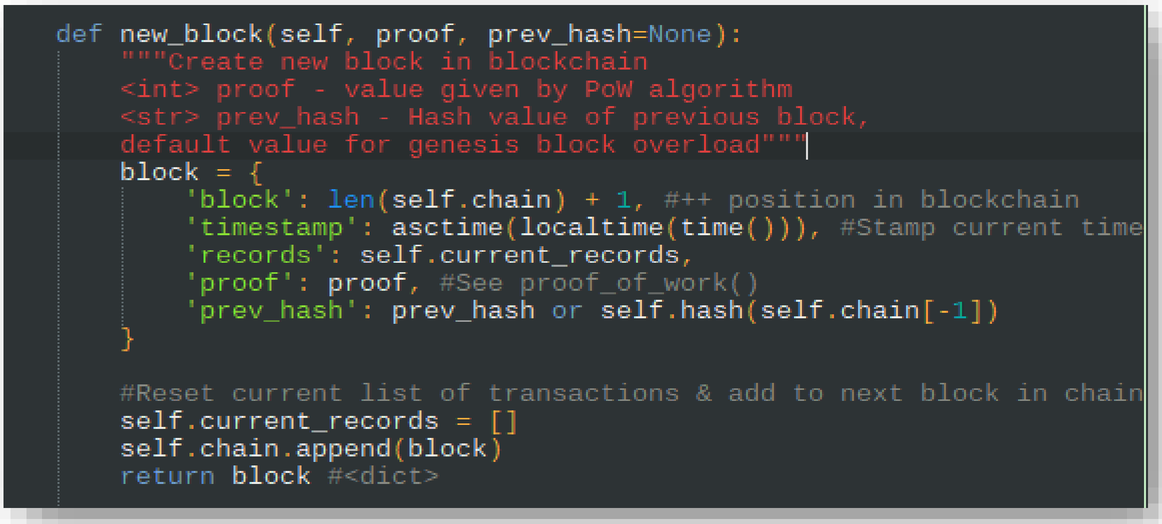
- **Constructor**: For the class constructor we need lists to store the blockchain itself and any pending current records, we also need to seed the blockchain by creating a genesis block (see new_block method).



```
class Blockchain(object):
    """Blockchain Class"""
    def __init__(self):
        #Constructor - Create initial lists to store blockchain info
        self.chain = [] #Stores entire blockchain
        self.current_records = [] #Temporary record storage
        self.new_block(prev_hash=1, proof=100) #Create genesis block
```

Figure 11: Blockchain __init__()

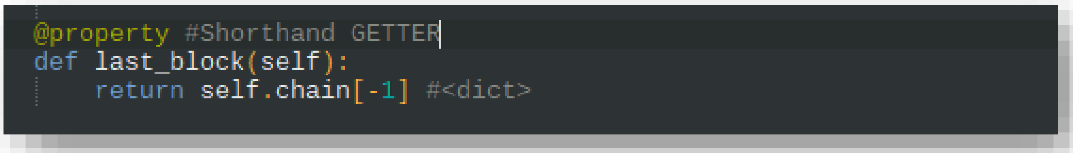
- **New Block**: Here we define block structure as well as how each block is appended to the chain. We provide a default None value for the prev_hash parameter for overloading when creating our genesis block (As it won't have a previous hash).



```
def new_block(self, proof, prev_hash=None):
    """Create new block in blockchain
    <int> proof - value given by PoW algorithm
    <str> prev_hash - Hash value of previous block,
    default value for genesis block overload"""
    block = {
        'block': len(self.chain) + 1, #++ position in blockchain
        'timestamp': asctime(localtime(time())), #Stamp current time
        'records': self.current_records,
        'proof': proof, #See proof_of_work()
        'prev_hash': prev_hash or self.hash(self.chain[-1])
    }

    #Reset current list of transactions & add to next block in chain
    self.current_records = []
    self.chain.append(block)
    return block #<dict>
```

Figure 12: Blockchain new_block()

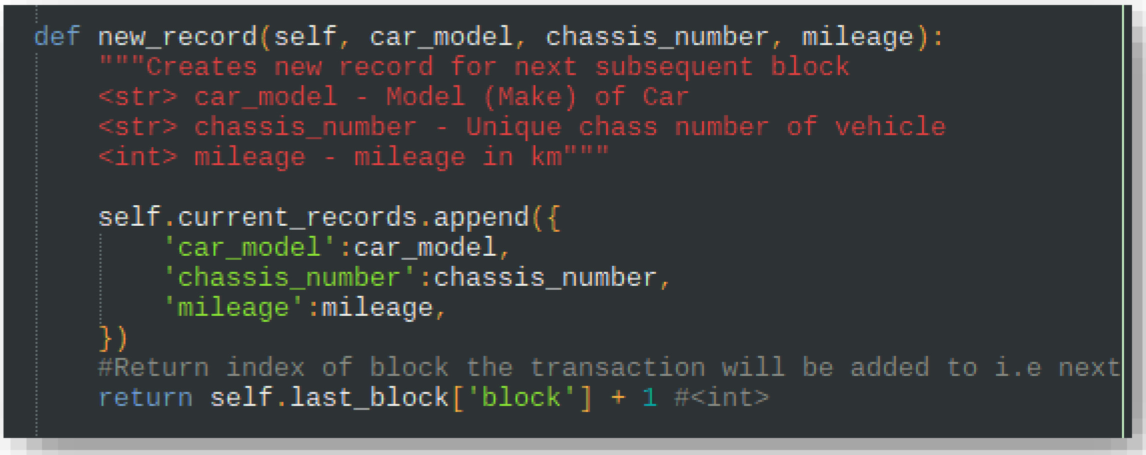
- **Last Block**: Method to return last block in chain, this will be used for getting specific info of last block throughout the program. The @property decorator in this context acts as a shorthand Getter function as we will be accessing last_block outside of the Blockchain class later on.



```
@property #Shorthand GETTER
def last_block(self):
    return self.chain[-1] #<dict>
```

Figure 13: Blockchain last_block()

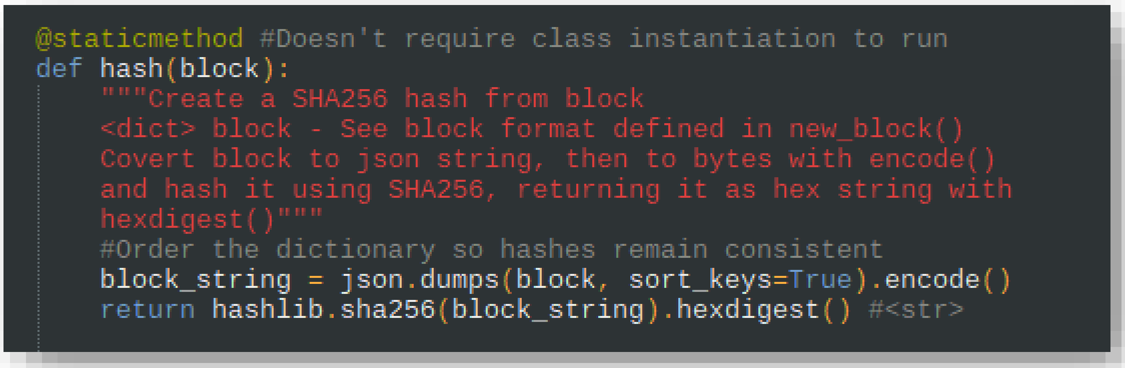
- **New Record**: Here we setup the record structure of the Blockchain. We create a dictionary instance variable and append it to the current_records list created in the constructor.



```python
def new_record(self, car_model, chassis_number, mileage):
    """Creates new record for next subsequent block
    <str> car_model - Model (Make) of Car
    <str> chassis_number - Unique chass number of vehicle
    <int> mileage - mileage in km"""

    self.current_records.append({
        'car_model':car_model,
        'chassis_number':chassis_number,
        'mileage':mileage,
    })
    #Return index of block the transaction will be added to i.e next
    return self.last_block['block'] + 1 #<int>
```

Figure 14: Blockchain new_record()

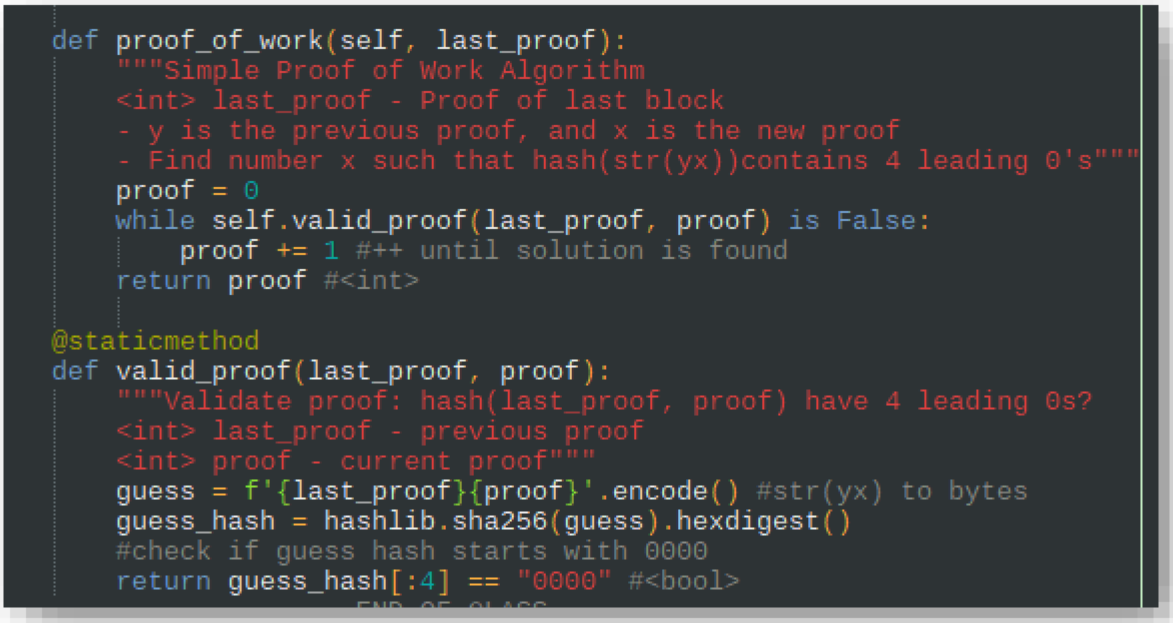
- **Hash**: We create a SHA256 hash from the block information converted to a JSON style string (Bytes). The @staticmethod decorator is used as this method doesn't modify any properties of the class itself but rather only operates locally on given parameters (in this case block).



```python
@staticmethod #Doesn't require class instantiation to run
def hash(block):
    """Create a SHA256 hash from block
    <dict> block - See block format defined in new_block()
    Covert block to json string, then to bytes with encode()
    and hash it using SHA256, returning it as hex string with
    hexdigest()"""
    #Order the dictionary so hashes remain consistent
    block_string = json.dumps(block, sort_keys=True).encode()
    return hashlib.sha256(block_string).hexdigest() #<str>
```

Figure 15: Blockchain hash()

- **Proof of Work & Valid Proof**: We have two methods for discerning the PoW proof value used for each block. In proof_of_work() we start with 0 for our proof value and run a check within valid_proof() to determine what it should ultimately equal, incrementing proof each time. In valid_proof() we hash this proof value along with the previous blocks proof value and check if the resulting output starts with four leading zeroes, if not we increment the current proof value and try again, if it does we return the proof value used. This essentially increases the time taken to create a new block so that new blocks cannot be easily spammed by any one entity. The difficulty can be increased or decreased depending on how many leading zeroes need to be matched.



```
def proof_of_work(self, last_proof):
    """Simple Proof of Work Algorithm
    <int> last_proof - Proof of last block
    - y is the previous proof, and x is the new proof
    - Find number x such that hash(str(yx))contains 4 leading 0's"""
    proof = 0
    while self.valid_proof(last_proof, proof) is False:
        proof += 1 #++ until solution is found
    return proof #<int>

@staticmethod
def valid_proof(last_proof, proof):
    """Validate proof: hash(last_proof, proof) have 4 leading 0s?
    <int> last_proof - previous proof
    <int> proof - current proof"""
    guess = f'{last_proof}{proof}'.encode() #str(yx) to bytes
    guess_hash = hashlib.sha256(guess).hexdigest()
    #check if guess hash starts with 0000
    return guess_hash[:4] == "0000" #<bool>
```

Figure 16: Blockchain proof_of_work() & valid_proof()

Program Functions

The global functions that make up the interface between the webapp and blockchain include:

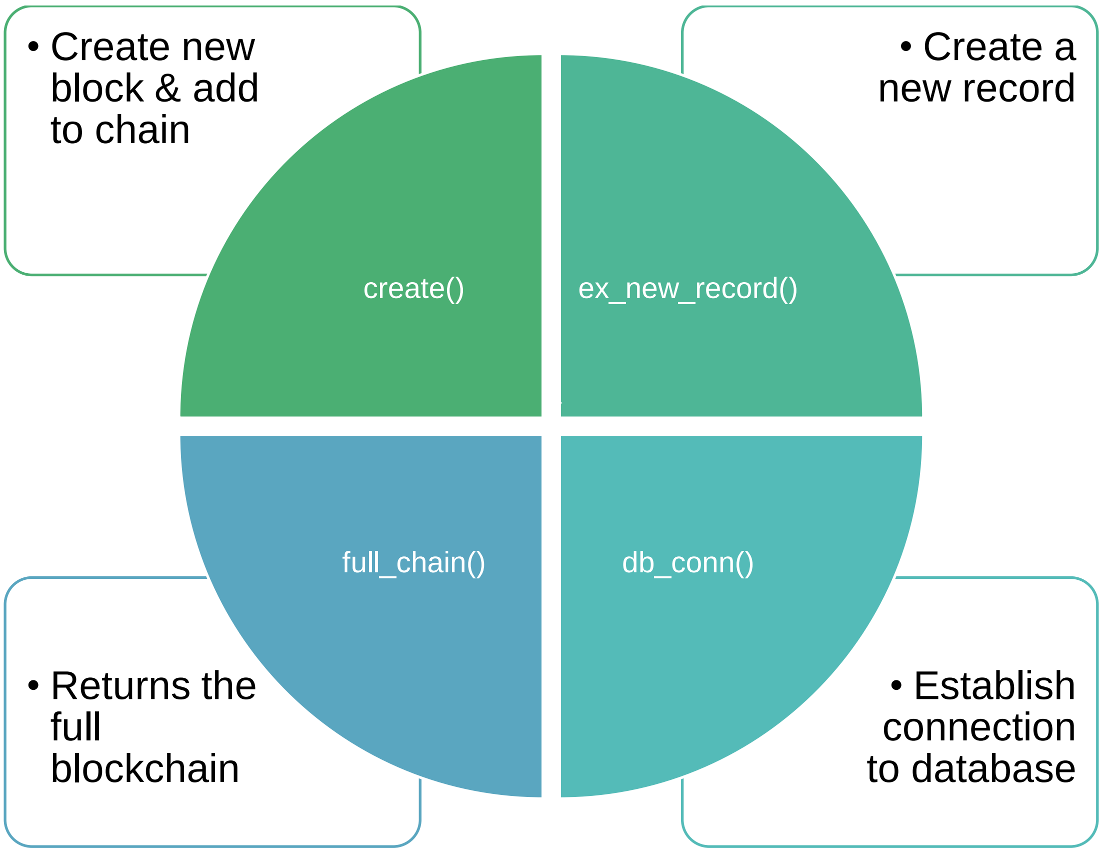
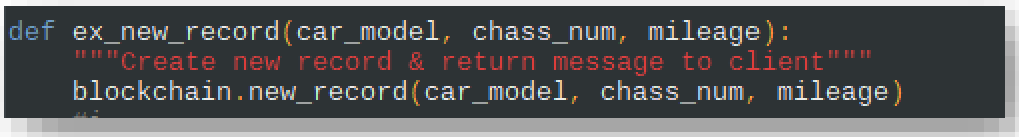

• Create new block & add to chain
create()
• Create a new record
ex_new_record()
full_chain()
• Returns the full blockchain
db_conn()
• Establish connection to database

Figure 17: Global Functions

**db_conn will be discussed in the next section.*

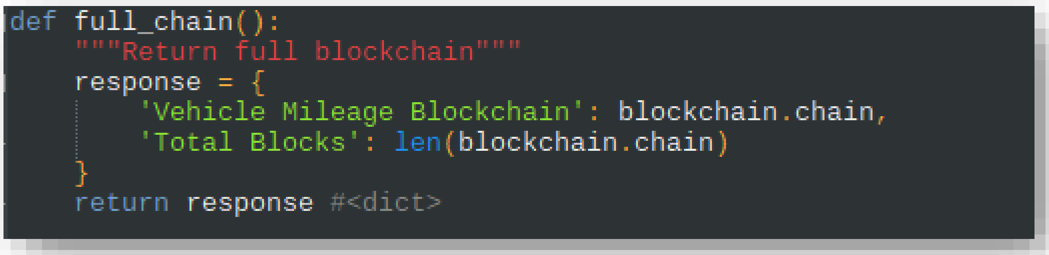
Ex_new_record(): Helper method for transferring data from user to blockchain.new_record().



```
def ex_new_record(car_model, chass_num, mileage):
    """Create new record & return message to client"""
    blockchain.new_record(car_model, chass_num, mileage)
```

Figure 18: ex_new_record()

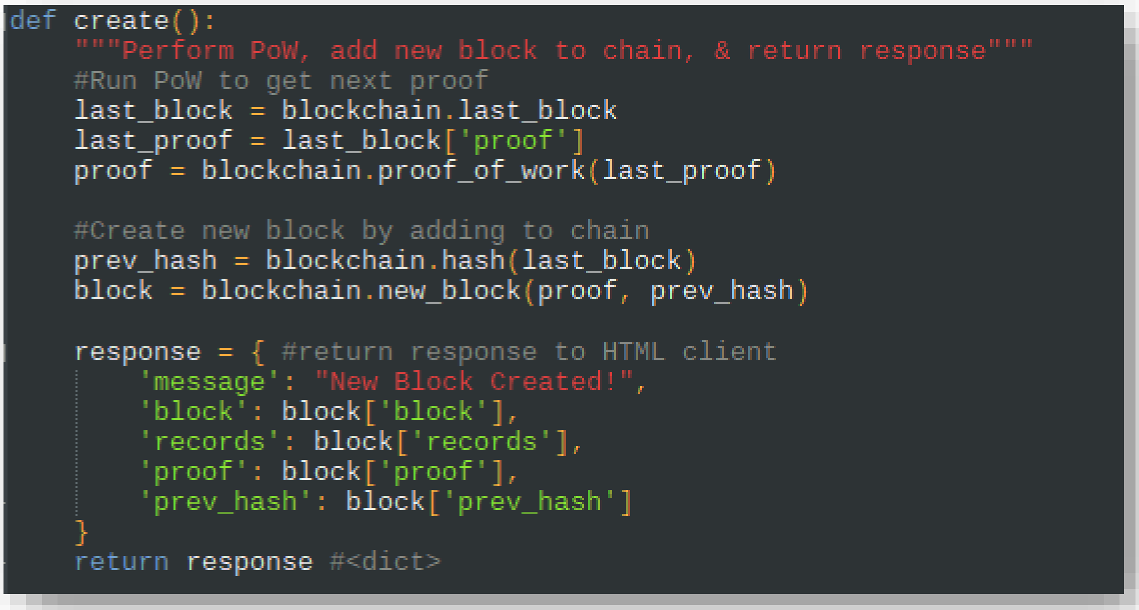
Full_chain(): Helper method to return full blockchain to web user.



```
def full_chain():
    """Return full blockchain"""
    response = {
        'Vehicle Mileage Blockchain': blockchain.chain,
        'Total Blocks': len(blockchain.chain)
    }
    return response #<dict>
```

Figure 19: full_blockchain()

Create(): Helper to create and add new block to the chain. We start by getting the most recent block in the chain, getting the value of its proof, and running the PoW algorithm to find a new proof value. We then get the hash of this previous block and send it along with our newly discovered proof value to the new_block() Blockchain class method. We also create a response aimed at the web client showing the information they added.



```
def create():
    """Perform PoW, add new block to chain, & return response"""
    #Run PoW to get next proof
    last_block = blockchain.last_block
    last_proof = last_block['proof']
    proof = blockchain.proof_of_work(last_proof)

    #Create new block by adding to chain
    prev_hash = blockchain.hash(last_block)
    block = blockchain.new_block(proof, prev_hash)

    response = { #return response to HTML client
        'message': "New Block Created!",
        'block': block['block'],
        'records': block['records'],
        'proof': block['proof'],
        'prev_hash': block['prev_hash']
    }
    return response #<dict>
```

Figure 20: create()

Database Functionality

We use the built-in Python library sqlite3 to create a simple user database based on a simple SQL schema script created. We create this database in a separate Python program called init_db.py.

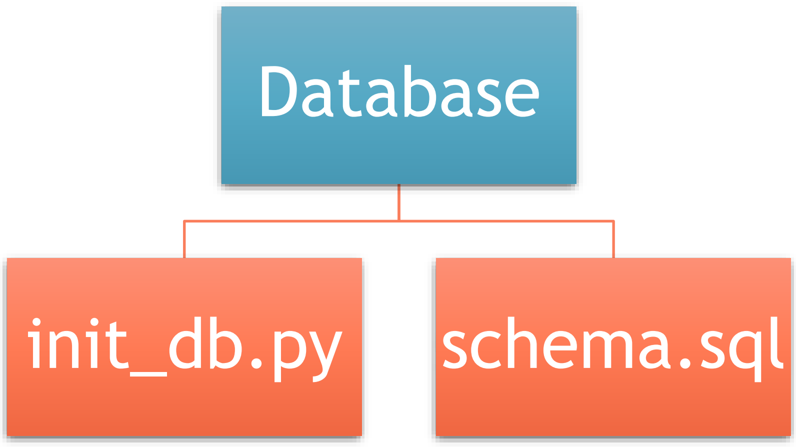
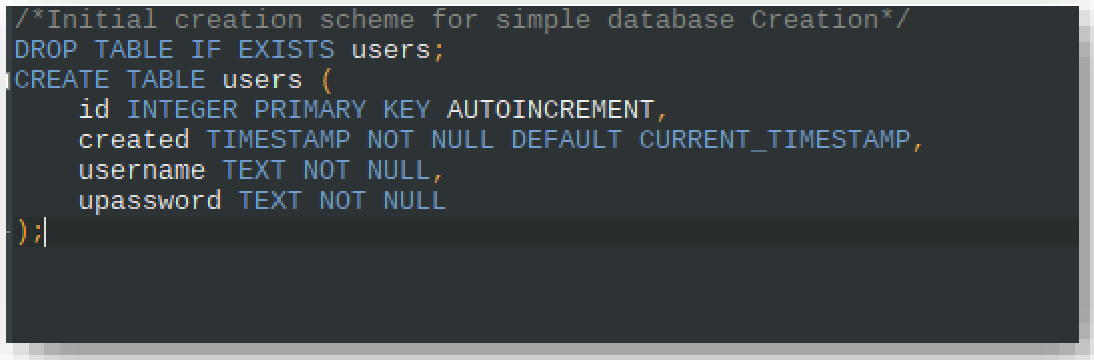


Figure 21: Components involved in database creation

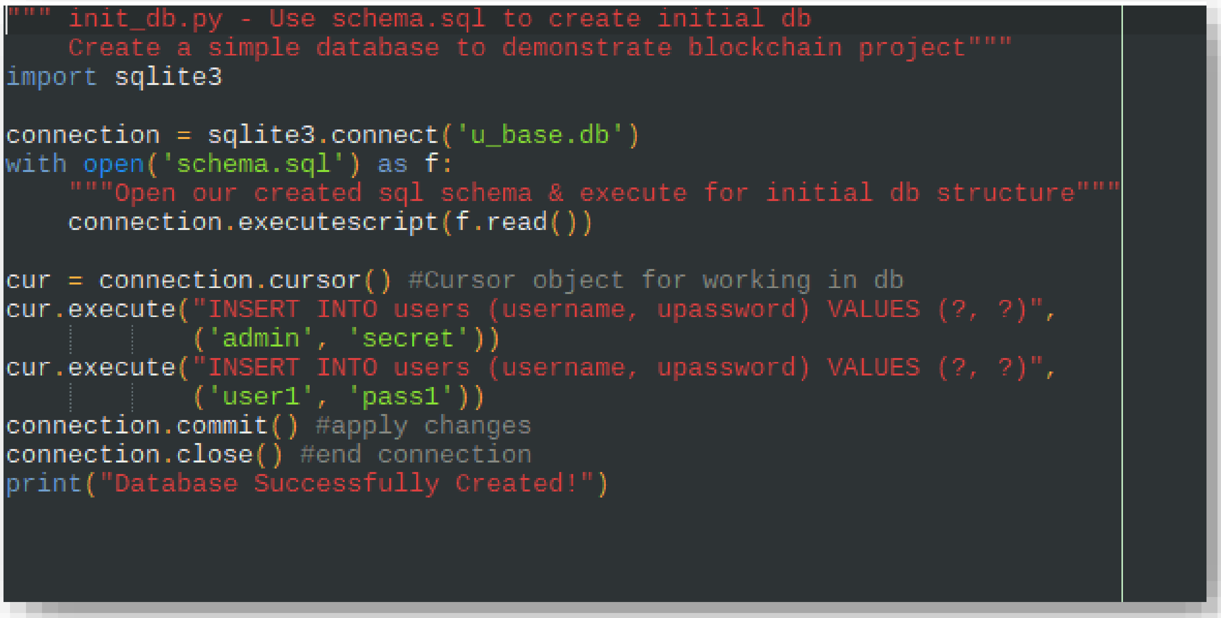
Schema.sql: This file contains a simple SQL script for creating the initial framework of our database. We create a users table, set the primary key to an auto incrementing integer value, add a timestamp, username (String), and upassword (String) fields.



```
/*Initial creation scheme for simple database Creation*/
DROP TABLE IF EXISTS users;
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    username TEXT NOT NULL,
    upassword TEXT NOT NULL
);
```

Figure 22: schema.sql

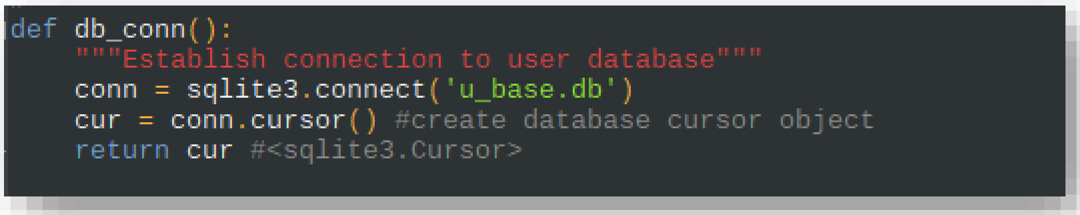
Init_db.py: We start off by importing the sqlite3 library and start a connection to target u_base.db (This will be created if it doesn't exist). We then open our created schema.sql file and execute its contents as a script with our connection target. We create a sqlite3 cursor object from the current connection so we can work within the database before inserting the details for two new user accounts into the users table. We finish by committing the changes made, closing the connection, and printing a message to the console.



```python
""" init_db.py - Use schema.sql to create initial db
    Create a simple database to demonstrate blockchain project"""
import sqlite3

connection = sqlite3.connect('u_base.db')
with open('schema.sql') as f:
    """Open our created sql schema & execute for initial db structure"""
    connection.executescript(f.read())

cur = connection.cursor() #Cursor object for working in db
cur.execute("INSERT INTO users (username, upassword) VALUES (?, ?)",
            ('admin', 'secret'))
cur.execute("INSERT INTO users (username, upassword) VALUES (?, ?)",
            ('user1', 'pass1'))
connection.commit() #apply changes
connection.close() #end connection
print("Database Successfully Created!")
```

Figure 23: init_db.py

Db_conn(): Back in blockchain_web_app.py we have a global db_conn() method for establishing a connection to our created u_base.db. This method returns a sqlite3 cursor object for further operations in our database.



```python
def db_conn():
    """Establish connection to user database"""
    conn = sqlite3.connect('u_base.db')
    cur = conn.cursor() #create database cursor object
    return cur #<sqlite3.Cursor>
```

Figure 24: db_conn()

Web Functions

For the web interface portion of the program we use a lightweight Python web API called Flask, this provides a lot of server-side functionality within Python, similar to PHP. We also use the Requests HTTP library; this will allow easy transfer of data between our Python program and HTML pages.

Here is a basic rundown of the web functions in our program (blue indicates user must be logged in to access).

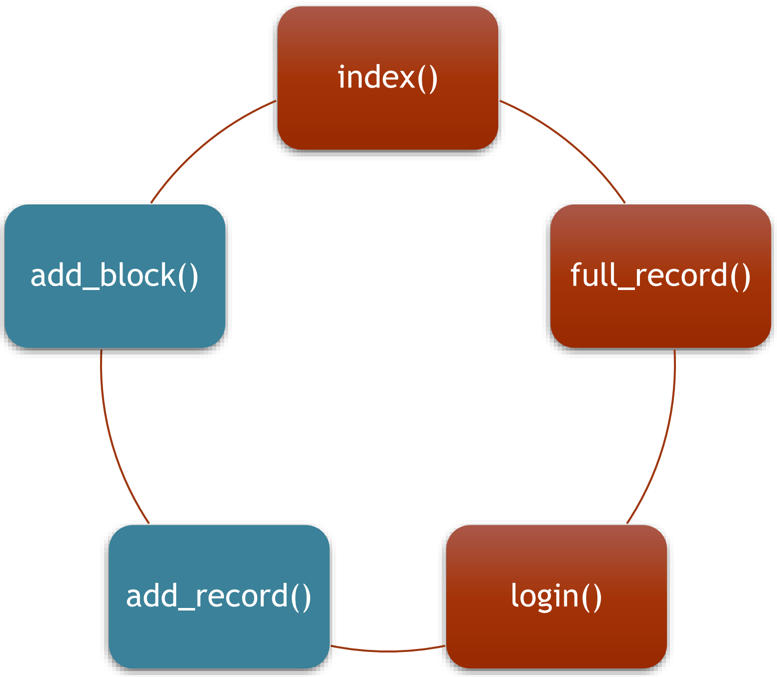
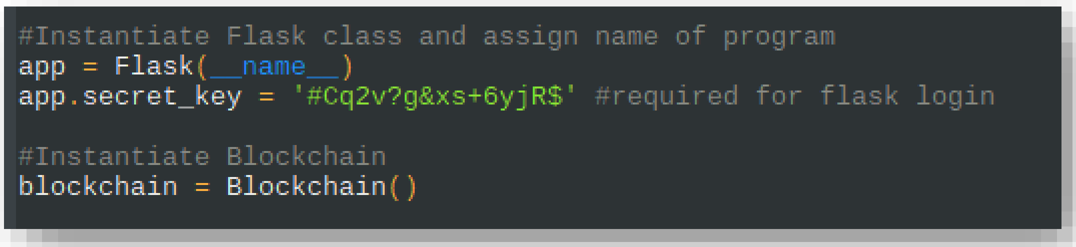


Figure 25: Overview of Web Functions

Before we setup our web functions we need to configure our Flask app (this is what essentially runs the webserver), we create an object of type Flask passing it the name of our program and setting up a secret key used for the login functionality. Since we are going to launch global functions from these web functions we will also instantiate a Blockchain object.

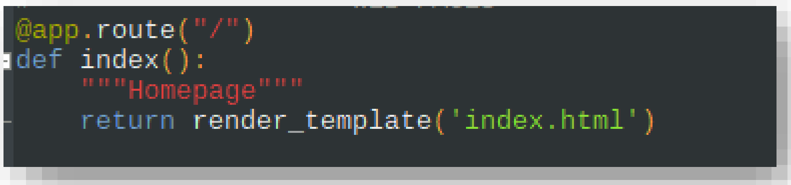


```
#Instantiate Flask class and assign name of program
app = Flask(__name__)
app.secret_key = '#Cq2v?g&xs+6yjR$' #required for flask login

#Instantiate Blockchain
blockchain = Blockchain()
```

Figure 26: Setting up Webapp

Index(): Simply renders our index.html page. With the Flask API we can work with structures known as "route" or "view" functions, these act similarly to the @property decorator but allow us to control functionality based on what resources (URI) of the Webapp a client accesses. The code below essentially means:

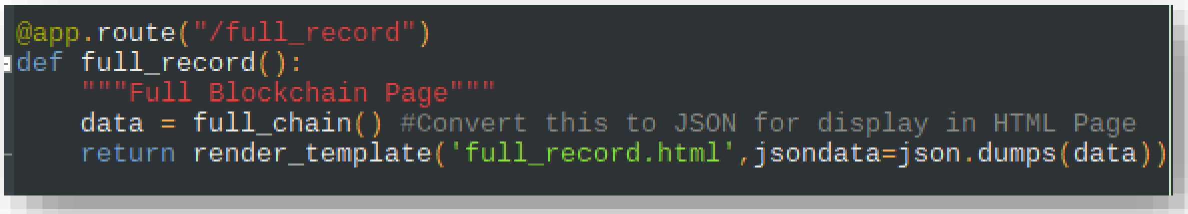
when the root (/) resource is accessed, run our index() function which in turn renders a HTML file.



```python
@app.route("/")
def index():
    """Homepage"""
    return render_template('index.html')
```

Figure 27: index()

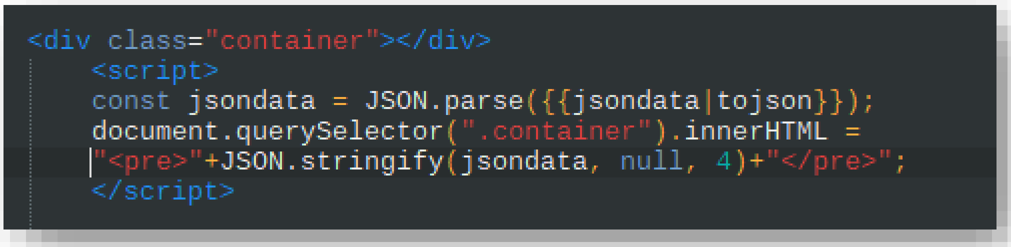
Full_record(): We define functionality for viewing the entire blockchain. We first run our full_chain() global function and then pass this data as a JSON style string to the full_record.html file for display.



```python
@app.route("/full_record")
def full_record():
    """Full Blockchain Page"""
    data = full_chain() #Convert this to JSON for display in HTML Page
    return render_template('full_record.html',jsondata=json.dumps(data))
```

Figure 28: full_record()

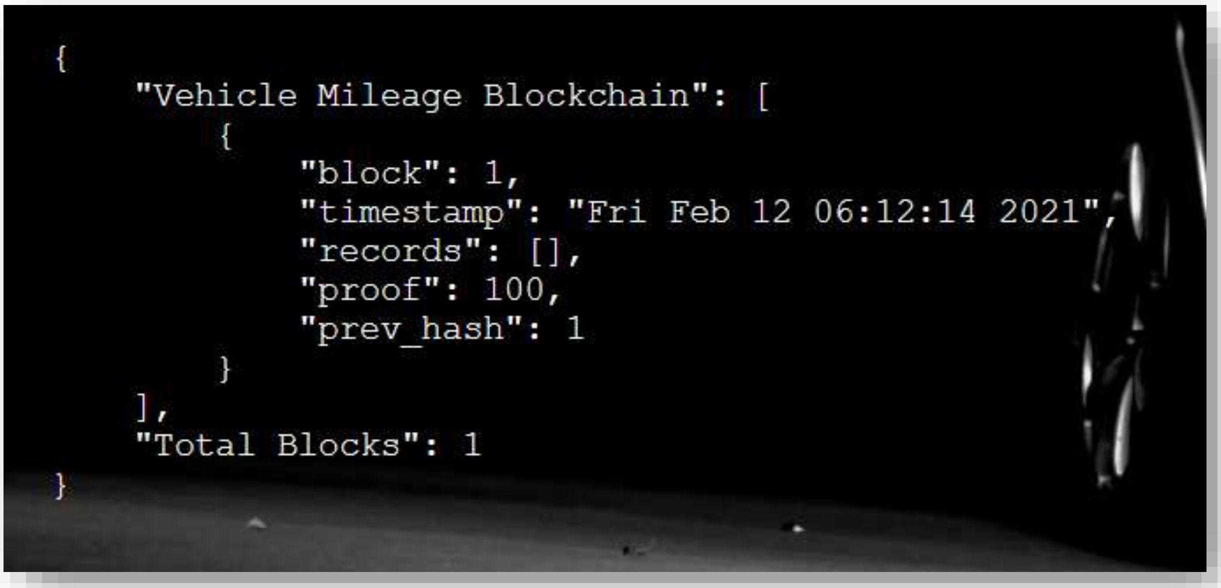
We display in our HTML file like such. We first parse the passed string using a tojson method before displaying it in the HTML element marked as "container" with JSON.stringify (formats the data nicely)



```html
<div class="container"></div>
    <script>
    const jsondata = JSON.parse({{jsondata|tojson}});
    document.querySelector(".container").innerHTML =
    "<pre>"+JSON.stringify(jsondata, null, 4)+"</pre>";
    </script>
```

Figure 29: full_record HTML code

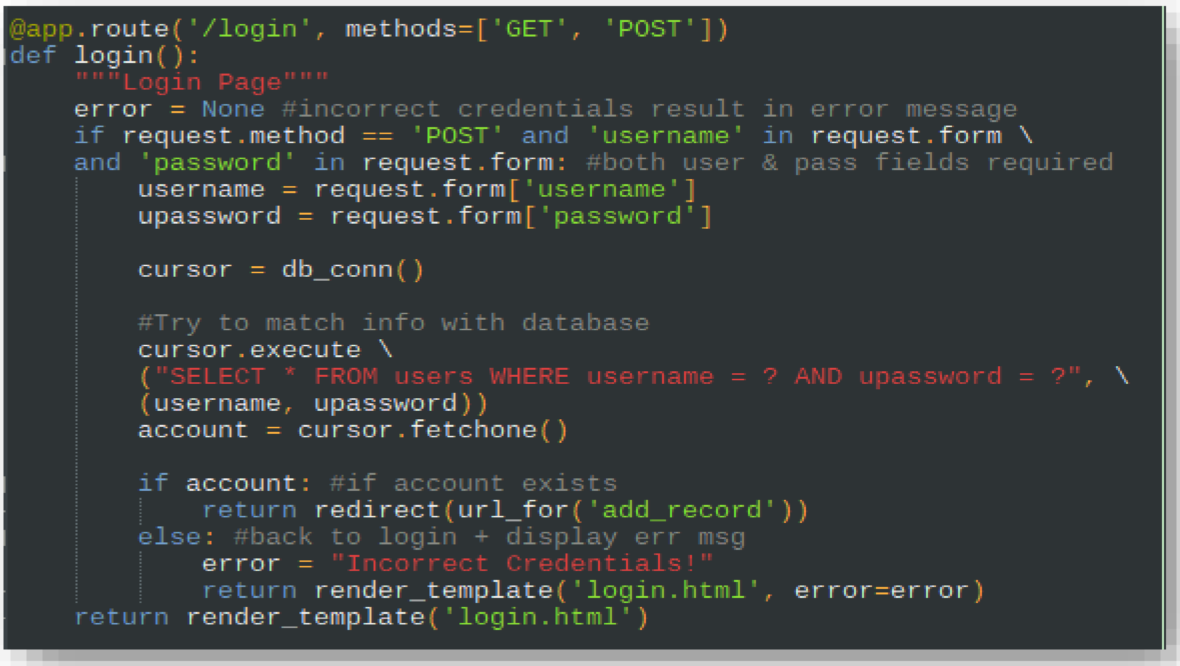
The end result is our data is displayed nicely on the webpage:



```
{
    "Vehicle Mileage Blockchain": [
        {
            "block": 1,
            "timestamp": "Fri Feb 12 06:12:14 2021",
            "records": [],
            "proof": 100,
            "prev_hash": 1
        }
    ],
    "Total Blocks": 1
}
```

Figure 30: Result of full_record()

Login(): On the login page we first check to see if both username & password fields are filled, if so we establish a connection to our database and check if the details entered match a user account, if this account exists the user is redirected to add_record.html. If any error occurs an error message is displayed and the user is redirected to the login page.



```
@app.route('/login', methods=['GET', 'POST'])
def login():
    """Login Page"""
    error = None #incorrect credentials result in error message
    if request.method == 'POST' and 'username' in request.form \
    and 'password' in request.form: #both user & pass fields required
        username = request.form['username']
        upassword = request.form['password']

        cursor = db_conn()

        #Try to match info with database
        cursor.execute \
        ("SELECT * FROM users WHERE username = ? AND upassword = ?", \
        (username, upassword))
        account = cursor.fetchone()

        if account: #if account exists
            return redirect(url_for('add_record'))
        else: #back to login + display err msg
            error = "Incorrect Credentials!"
            return render_template('login.html', error=error)
    return render_template('login.html')
```

Figure 31: login()

HTML login page:

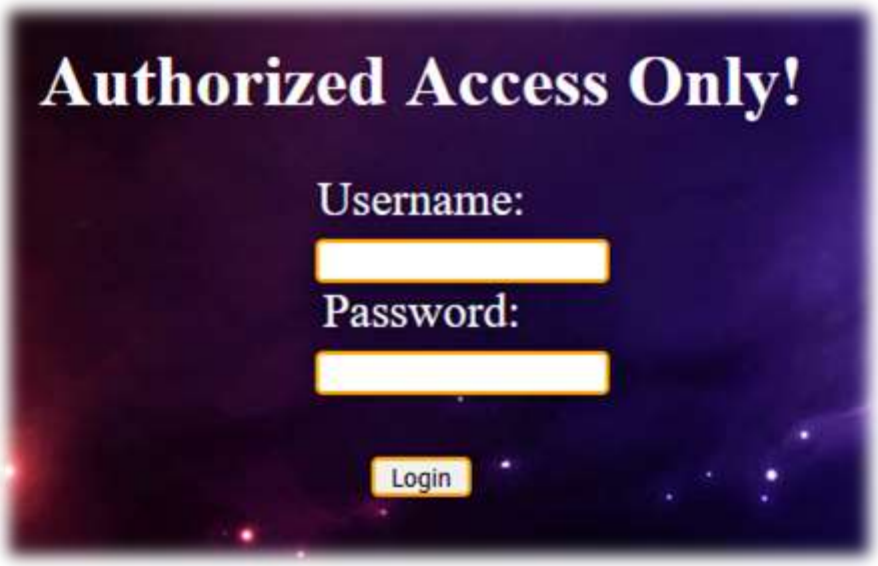
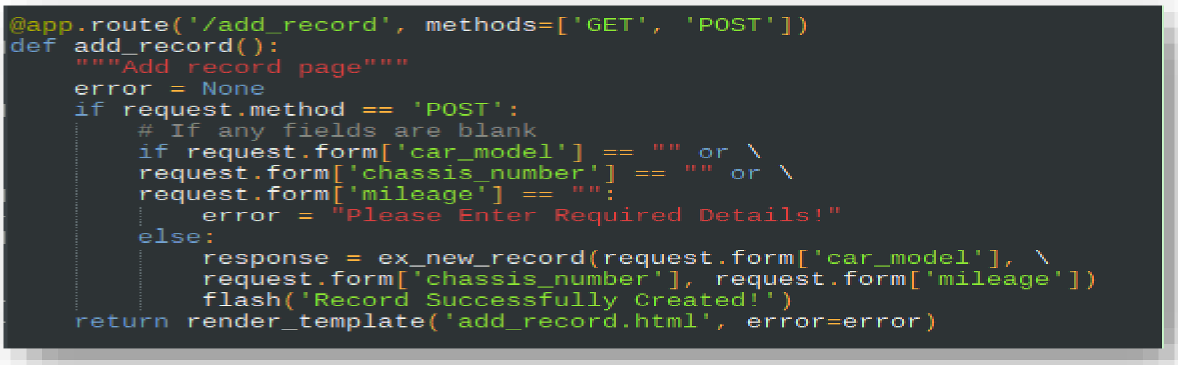


Figure 32: login screen

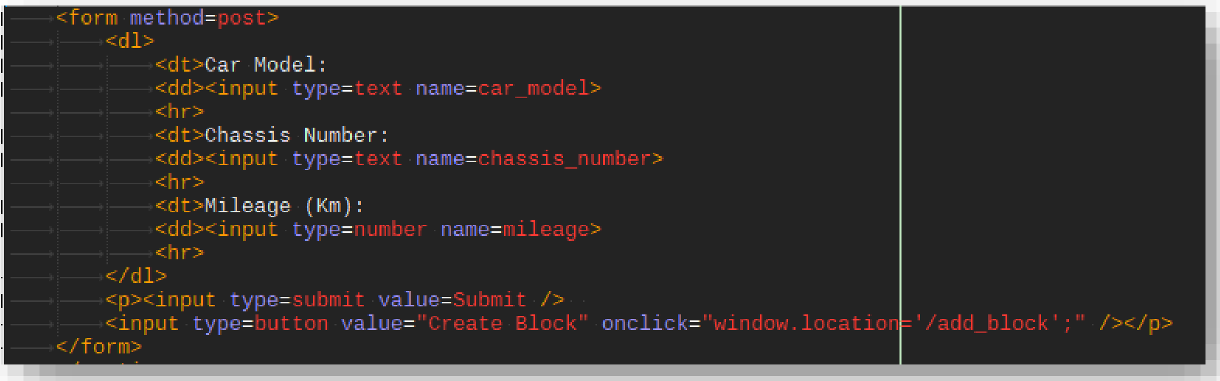
Add_record(): Here we try and parse record information POSTed by HTML form data, we make sure none of the fields are left blank and display an error message if so. If all required info is present we supply this info to the ex_new_record() global function and flash a success message to screen.



```
@app.route('/add_record', methods=['GET', 'POST'])
def add_record():
    """Add record page"""
    error = None
    if request.method == 'POST':
        # If any fields are blank
        if request.form['car_model'] == "" or \
        request.form['chassis_number'] == "" or \
        request.form['mileage'] == "":
            error = "Please Enter Required Details!"
        else:
            response = ex_new_record(request.form['car_model'], \
            request.form['chassis_number'], request.form['mileage'])
            flash('Record Successfully Created!')
    return render_template('add_record.html', error=error)
```

Figure 33: add_record()

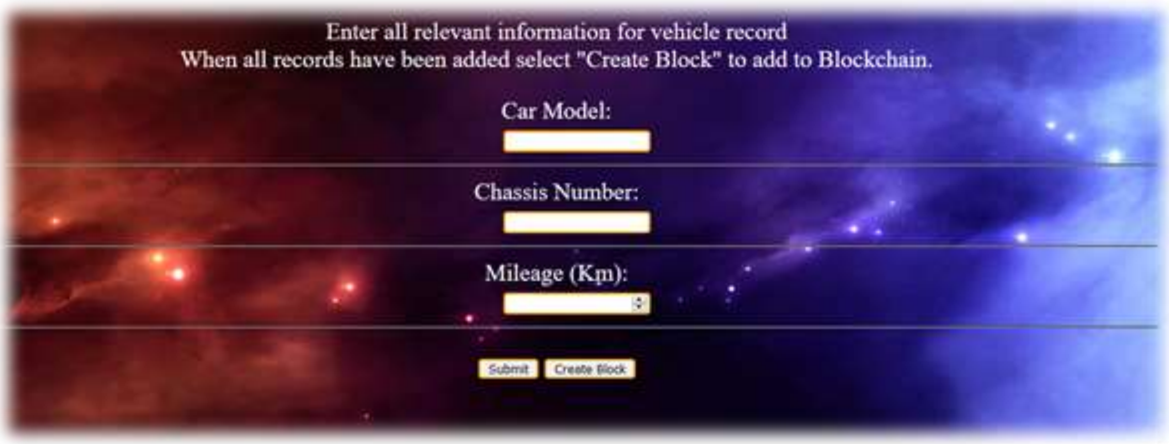
Structure of the HTML form:



```
    <form method=post>
        <dl>
            <dt>Car Model:
            <dd><input type=text name=car_model>
            <hr>
            <dt>Chassis Number:
            <dd><input type=text name=chassis_number>
            <hr>
            <dt>Mileage (Km):
            <dd><input type=number name=mileage>
            <hr>
        </dl>
        <p><input type=submit value=Submit />
        <input type=button value="Create Block" onclick="window.location='/add_block';" /></p>
    </form>
```

Figure 34: add_record HTML form

Web form:



Enter all relevant information for vehicle record
When all records have been added select "Create Block" to add to Blockchain.

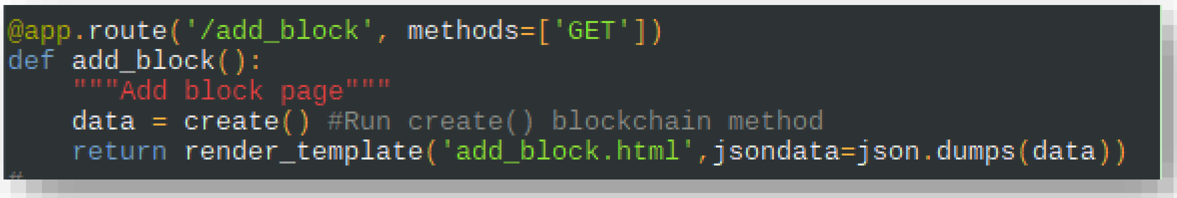
Car Model:

Chassis Number:

Mileage (Km):

Figure 35: add record form

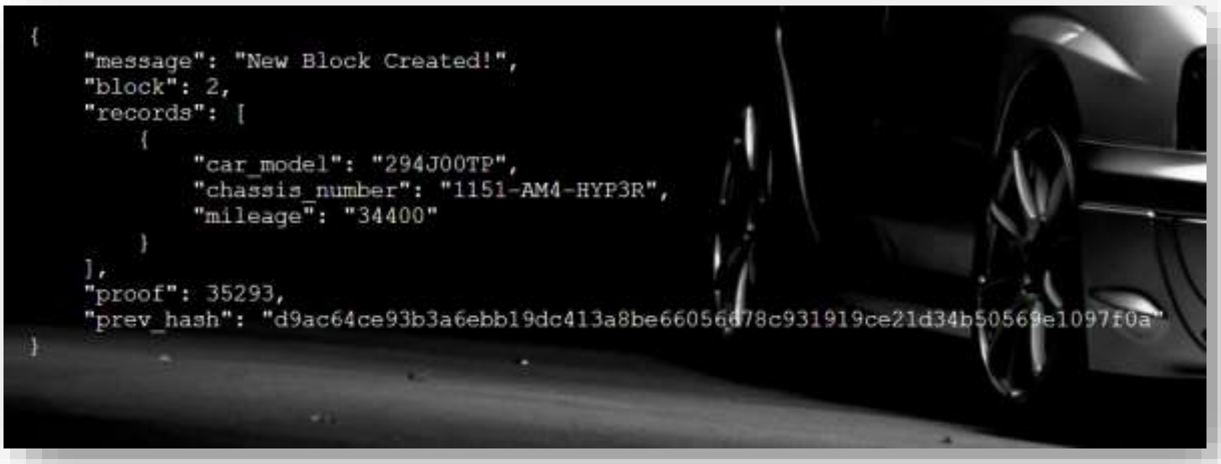
Add_block(): Here we run the create() global function to add all temporary records added to the current block in add_record() and append a new block to the blockchain. We also send this data to the rendered add_block.html page.



```
@app.route('/add_block', methods=['GET'])
def add_block():
    """Add block page"""
    data = create() #Run create() blockchain method
    return render_template('add_block.html',jsondata=json.dumps(data))
```

Figure 36: add_block()

The displayed data on the HTML page is in the format:



```
{
    "message": "New Block Created!",
    "block": 2,
    "records": [
        {
            "car_model": "294J00TP",
            "chassis_number": "1151-AM4-HYP3R",
            "mileage": "34400"
        }
    ],
    "proof": 35293,
    "prev_hash": "d9ac64ce93b3a6ebb19dc413a8be66056678c931919ce21d34b50569e1097f0a"
}
```

Figure 37: add_block HTML result

Background Network: REDACTED had a great idea to create a hypothetical background network that this project would be hosted on, he created a diagram of it using Packet Tracer:

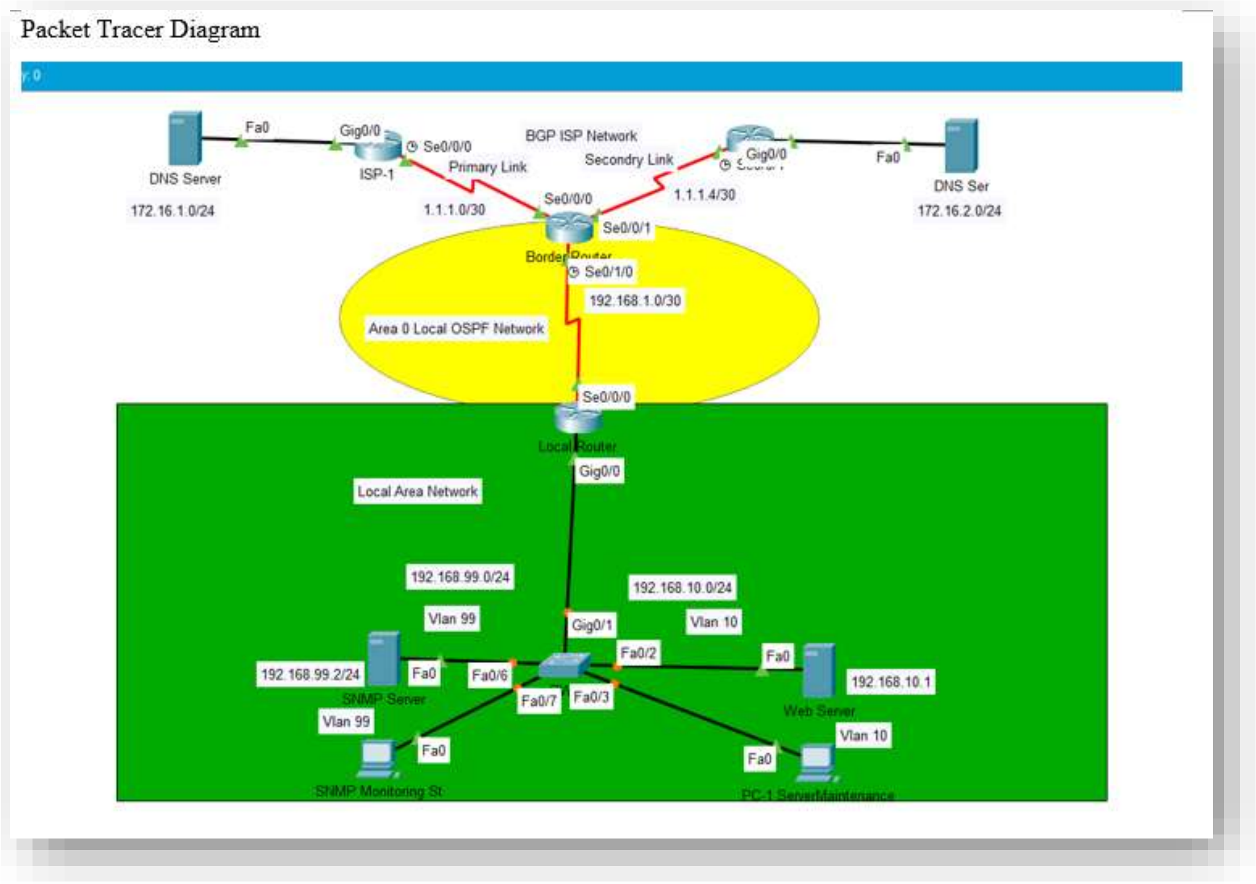
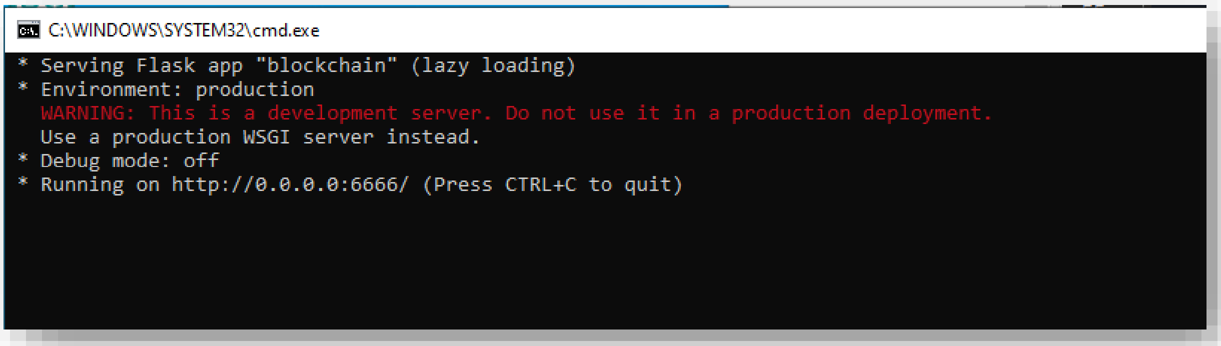


Figure 38: Network Diagram

Testing & Troubleshooting

Initial blockchain test: I initially tested out the blockchain functionality without any website involvement.

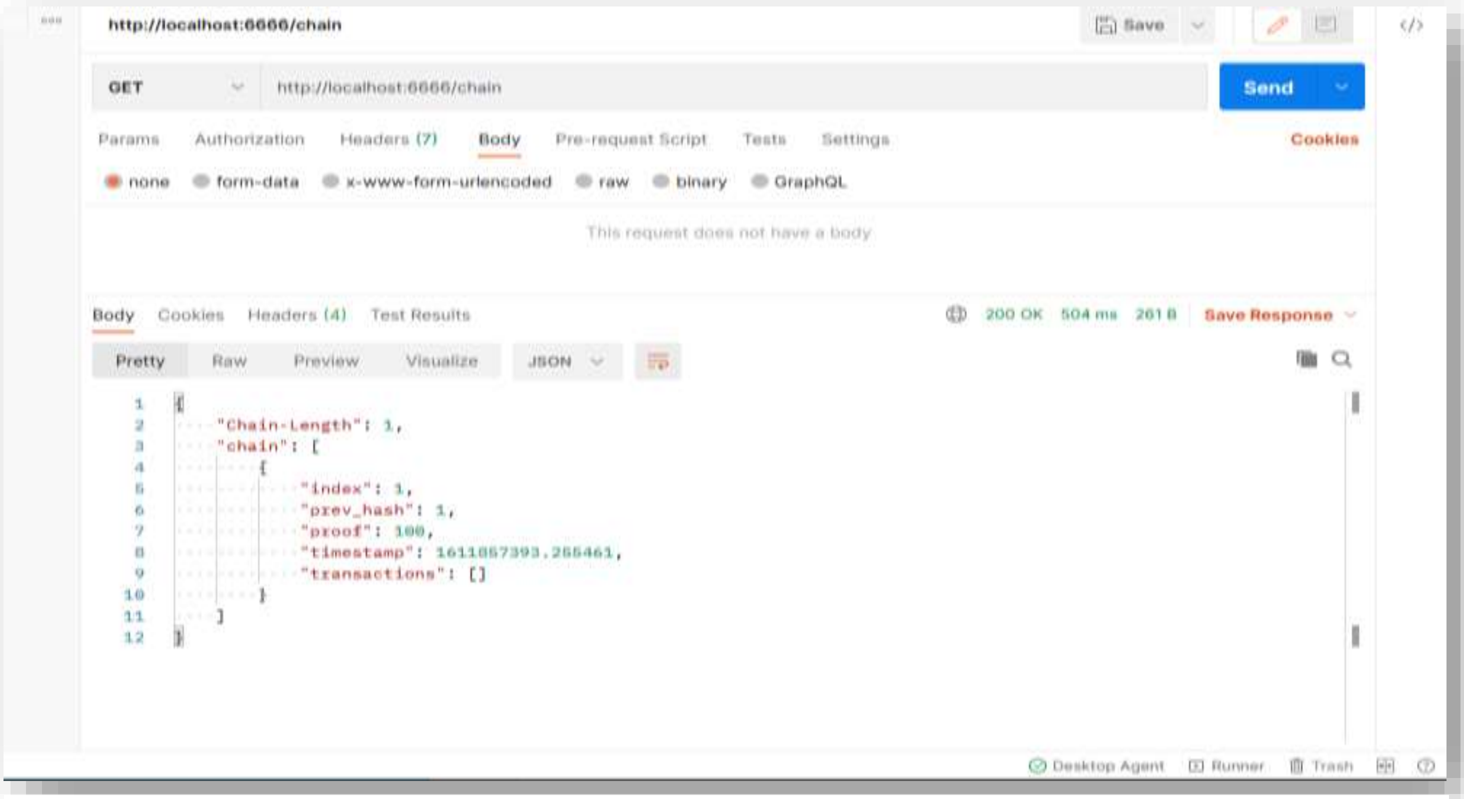
I run the blockchain_flask_app.py program which causes the Flask server to start:



```
* Serving Flask app "blockchain" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:6666/ (Press CTRL+C to quit)
```

Figure 39: init test 1

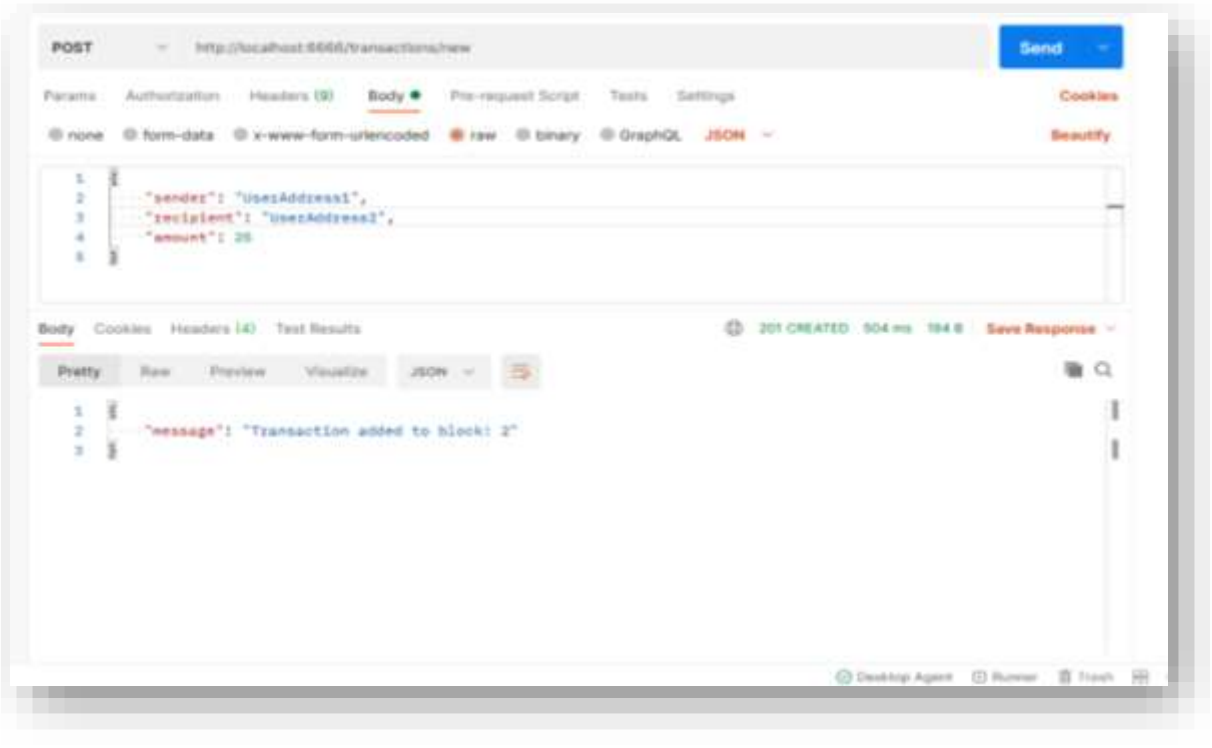
I used an app called Postman in the initial testing, it's a framework for API development, however in this context I used it to test out the functionality of our blockchain program through the use of HTTP GET & POST requests. I initially got the starting blockchain, here we can also see the implicitly created genesis block.



```
{
    "Chain-Length": 1,
    "chain": [
        {
            "index": 1,
            "prev_hash": 1,
            "proof": 100,
            "timestamp": 1611857393.255461,
            "transactions": []
        }
    ]
}
```

Figure 40: init test 2

I then created two new records by POSTing the relevant information through HTTP, we can see they were both added to block 2.

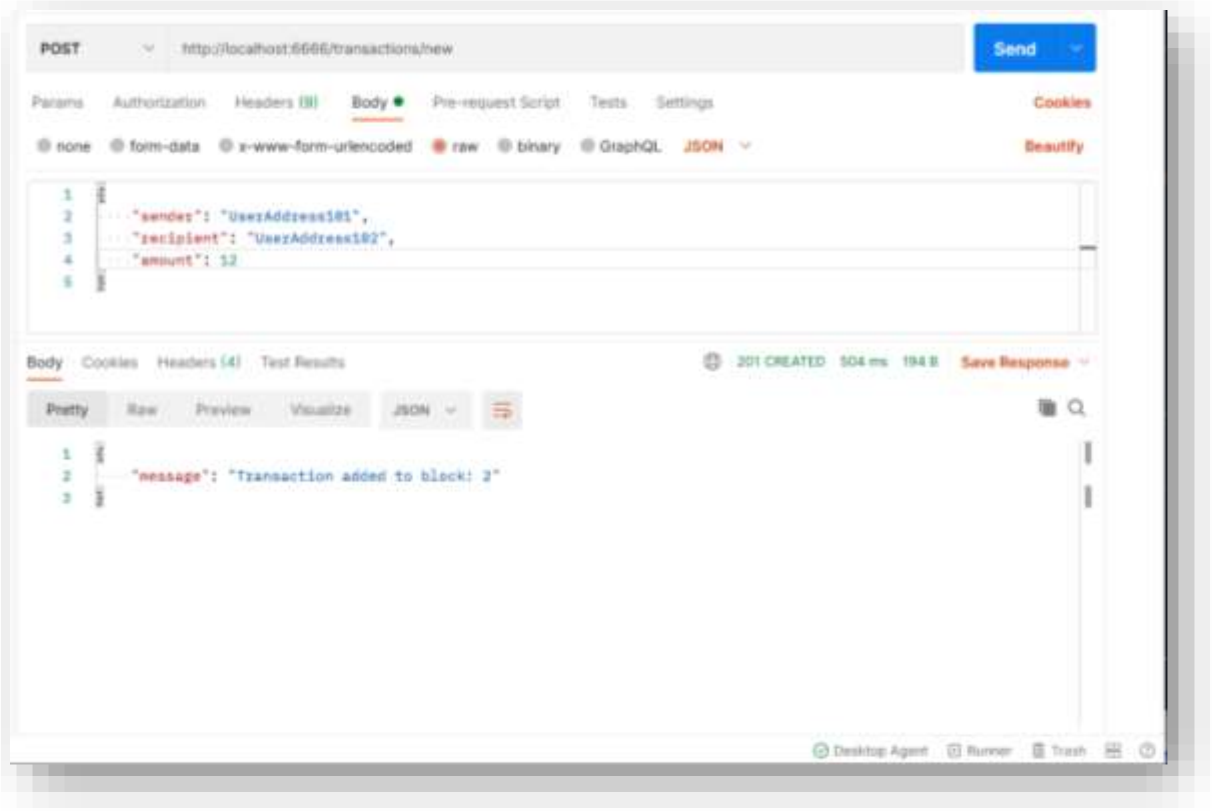

POST http://localhost:6666/transactions/new

```
{
    "sender": "UserAddress1",
    "recipient": "UserAddress2",
    "amount": 25
}
```

201 CREATED 504 ms 194 B

```
{
    "message": "Transaction added to block: 2"
}
```

Figure 41: init test 3


POST http://localhost:6666/transactions/new

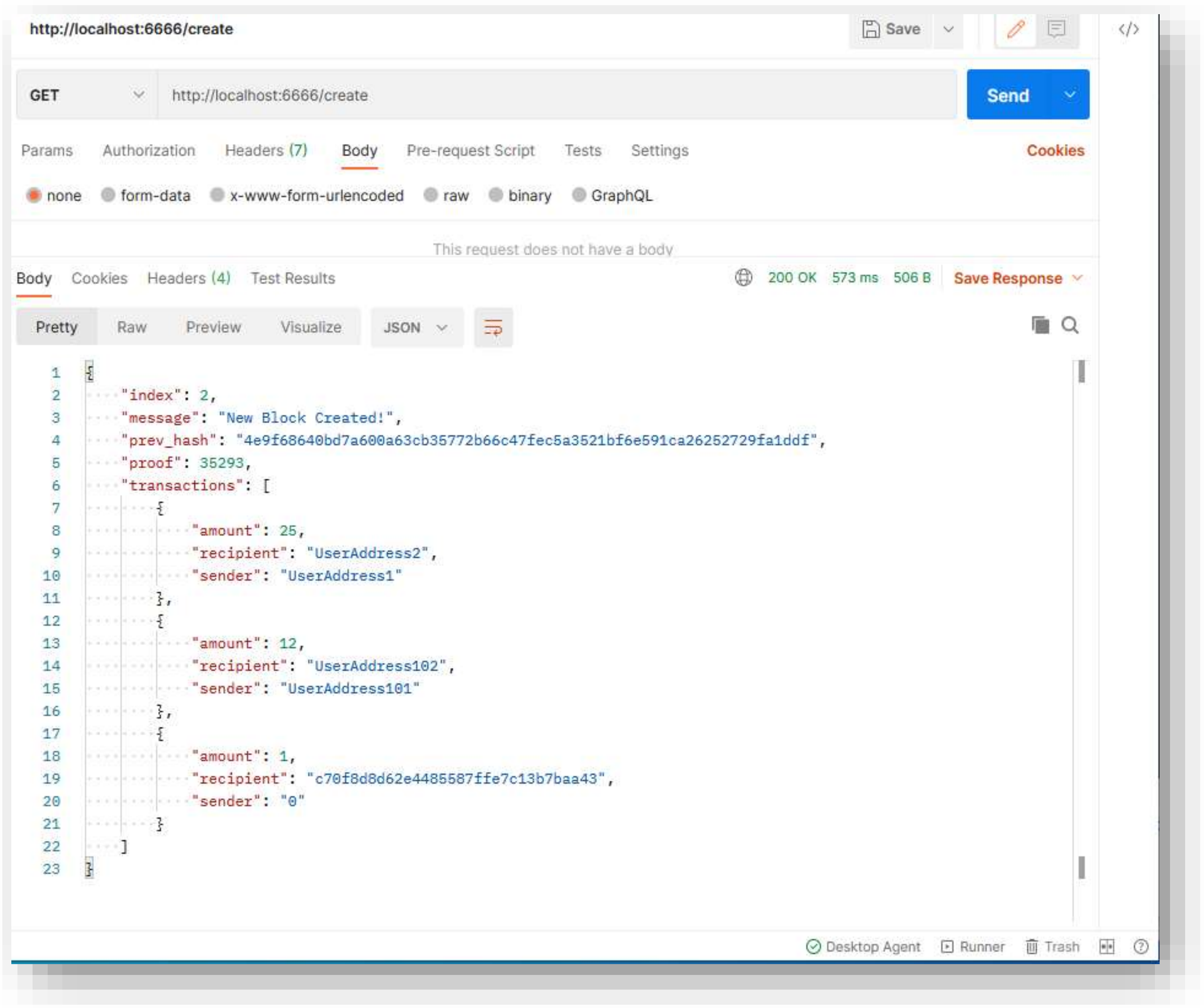
```
{
    "sender": "UserAddress101",
    "recipient": "UserAddress102",
    "amount": 12
}
```

201 CREATED 504 ms 194 B

```
{
    "message": "Transaction added to block: 2"
}
```

Figure 42: init test 4

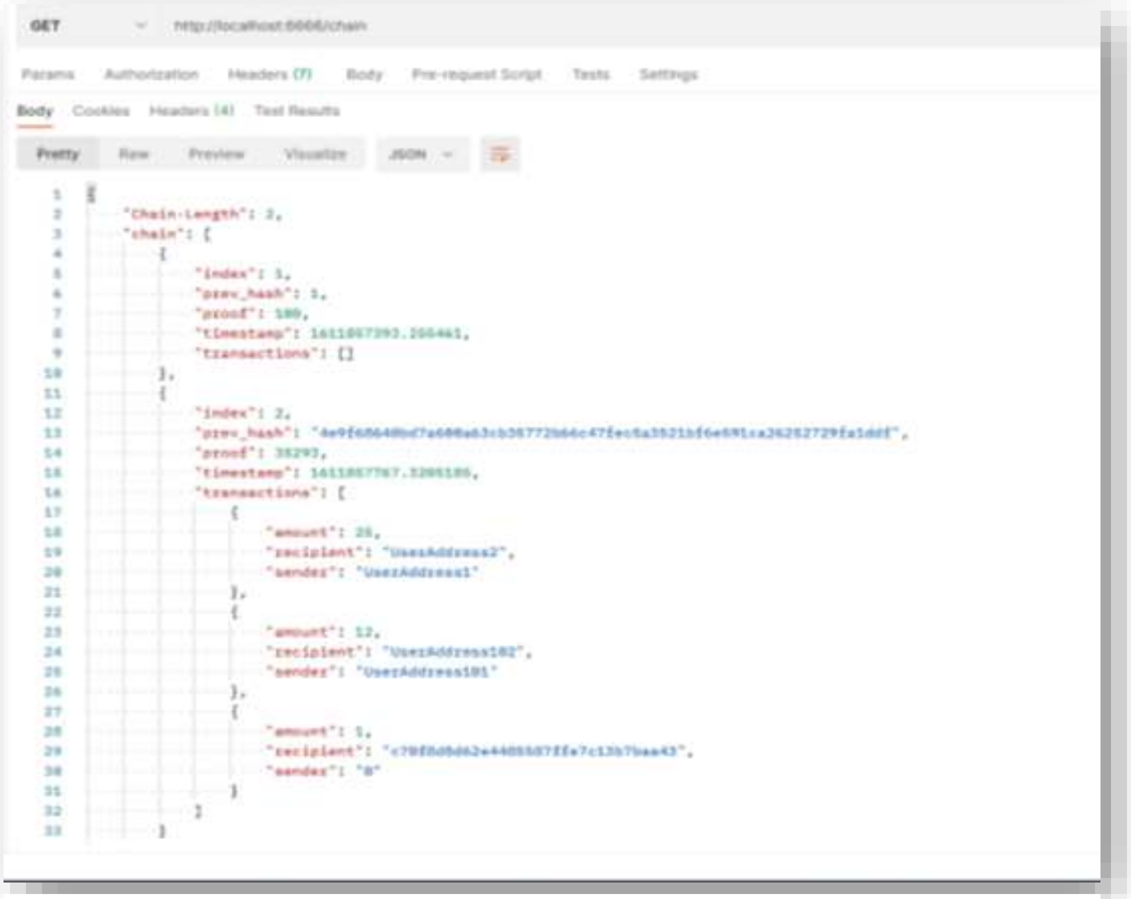
We then have to create the new block in order to add it to the blockchain. *Note:* the last transaction here (“sender”: 0) is part of a functionality that has since been removed.



```
{
    "index": 2,
    "message": "New Block Created!",
    "prev_hash": "4e9f68640bd7a600a63cb35772b66c47fec5a3521bf6e591ca26252729fa1ddf",
    "proof": 35293,
    "transactions": [
        {
            "amount": 25,
            "recipient": "UserAddress2",
            "sender": "UserAddress1"
        },
        {
            "amount": 12,
            "recipient": "UserAddress102",
            "sender": "UserAddress101"
        },
        {
            "amount": 1,
            "recipient": "c70f8d8d62e4485587ffe7c13b7baa43",
            "sender": "0"
        }
    ]
}
```

Figure 43: init test 5

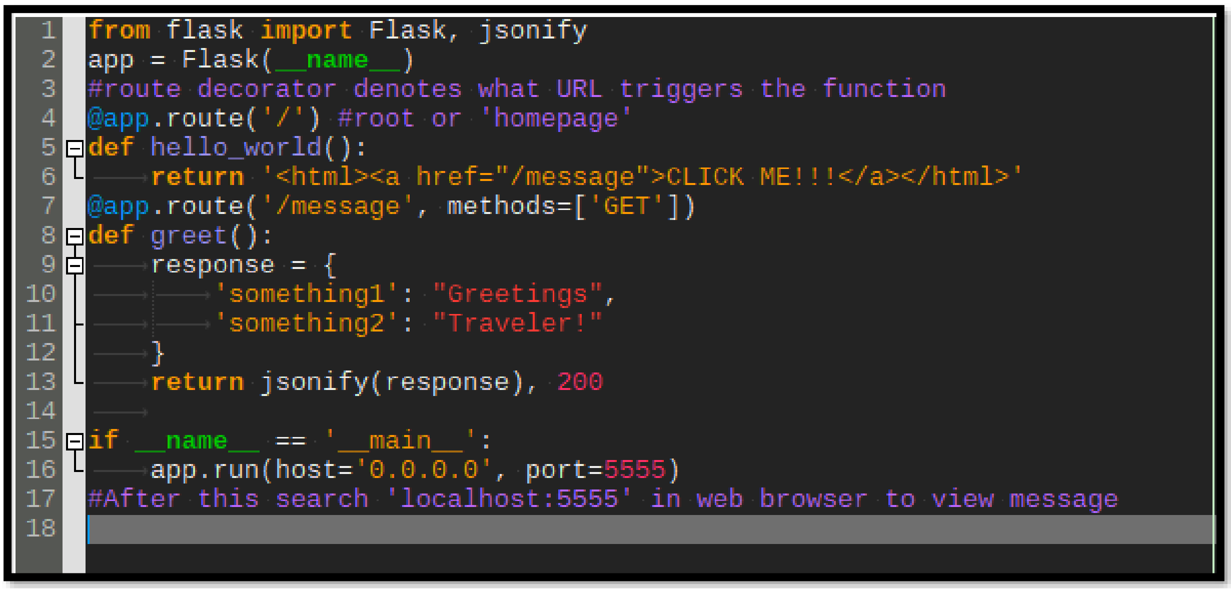
We can then view the entire blockchain to see our newly added block



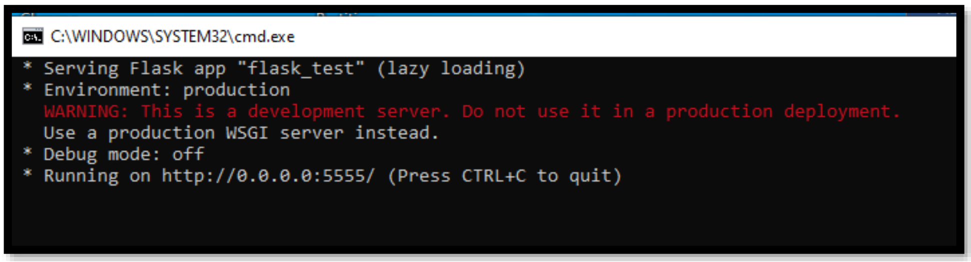
```
{
    "Chain-Length": 2,
    "chain": [
        {
            "index": 1,
            "prev_hash": 1,
            "proof": 100,
            "timestamp": 1611857393.266461,
            "transactions": []
        },
        {
            "index": 2,
            "prev_hash": "4e9f6864 0bd7a680ab3cb38772b66c47fec6a3521bf6e691ca26282729fa1ddf",
            "proof": 35293,
            "timestamp": 1611857767.3205186,
            "transactions": [
                {
                    "amount": 25,
                    "recipient": "UserAddress2",
                    "sender": "UserAddress1"
                },
                {
                    "amount": 12,
                    "recipient": "UserAddress102",
                    "sender": "UserAddress101"
                },
                {
                    "amount": 1,
                    "recipient": "c79f8d0d62e4405587ffe7c13b7baa43",
                    "sender": "0"
                }
            ]
        }
```

Figure 44: init test 6

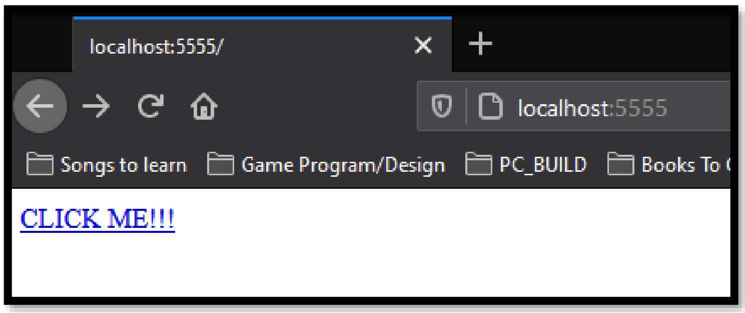
Flask test: My initial Flask test, here I setup and test basic flask functionality.



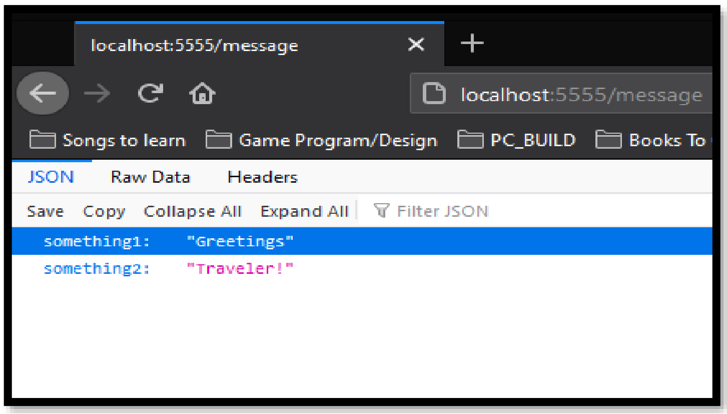
```python
from flask import Flask, jsonify
app = Flask(__name__)
#route decorator denotes what URL triggers the function
@app.route('/') #root or 'homepage'
def hello_world():
    return '<html><a href="/message">CLICK ME!!!</a></html>'
@app.route('/message', methods=['GET'])
def greet():
    response = {
        'something1': "Greetings",
        'something2': "Traveler!"
    }
    return jsonify(response), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5555)
#After this search 'localhost:5555' in web browser to view message
```



```
C:\WINDOWS\SYSTEM32\cmd.exe
 * Serving Flask app "flask_test" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5555/ (Press CTRL+C to quit)
```



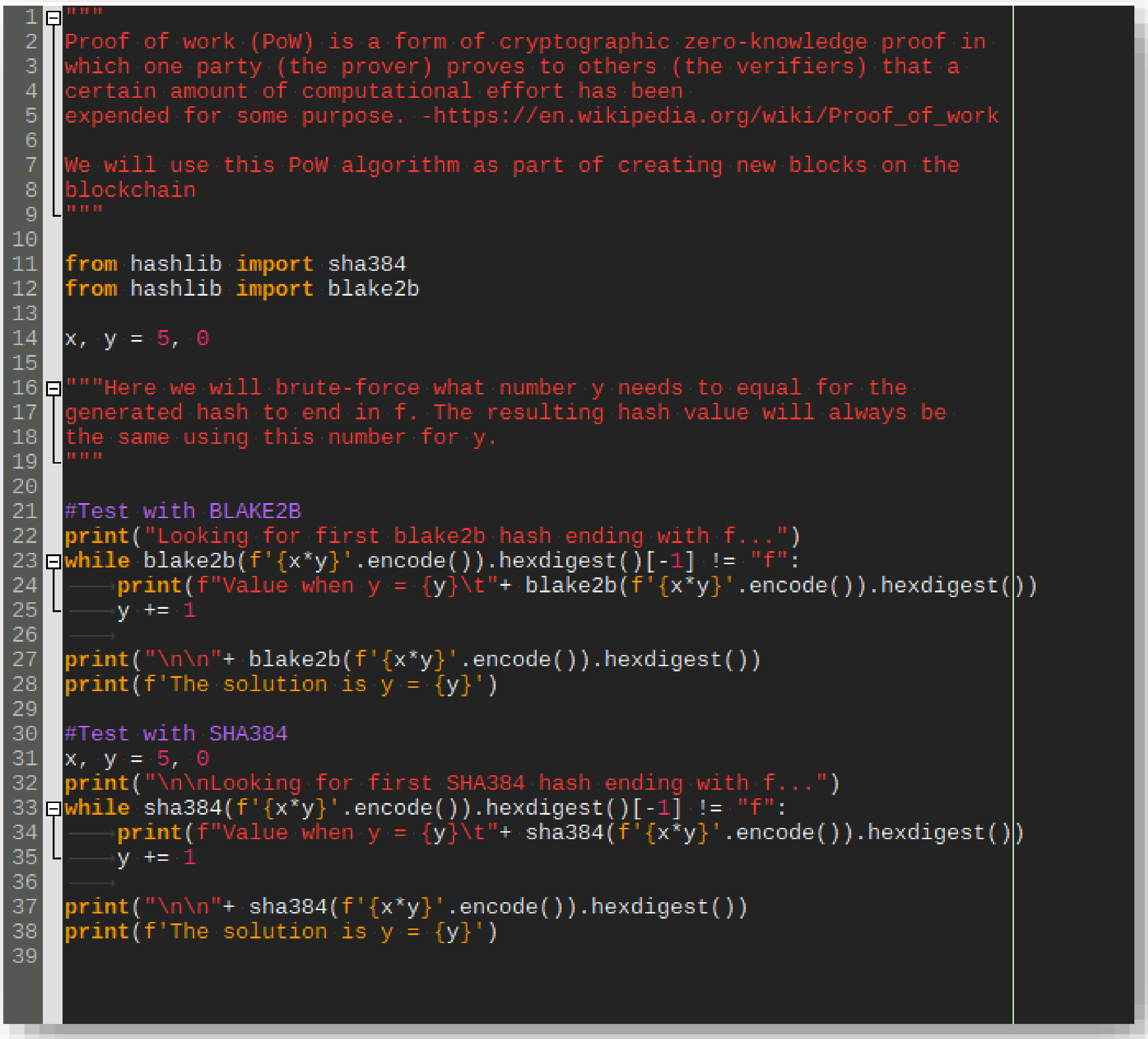
CLICK ME!!!



```
something1:    "Greetings"
something2:    "Traveler!"
```

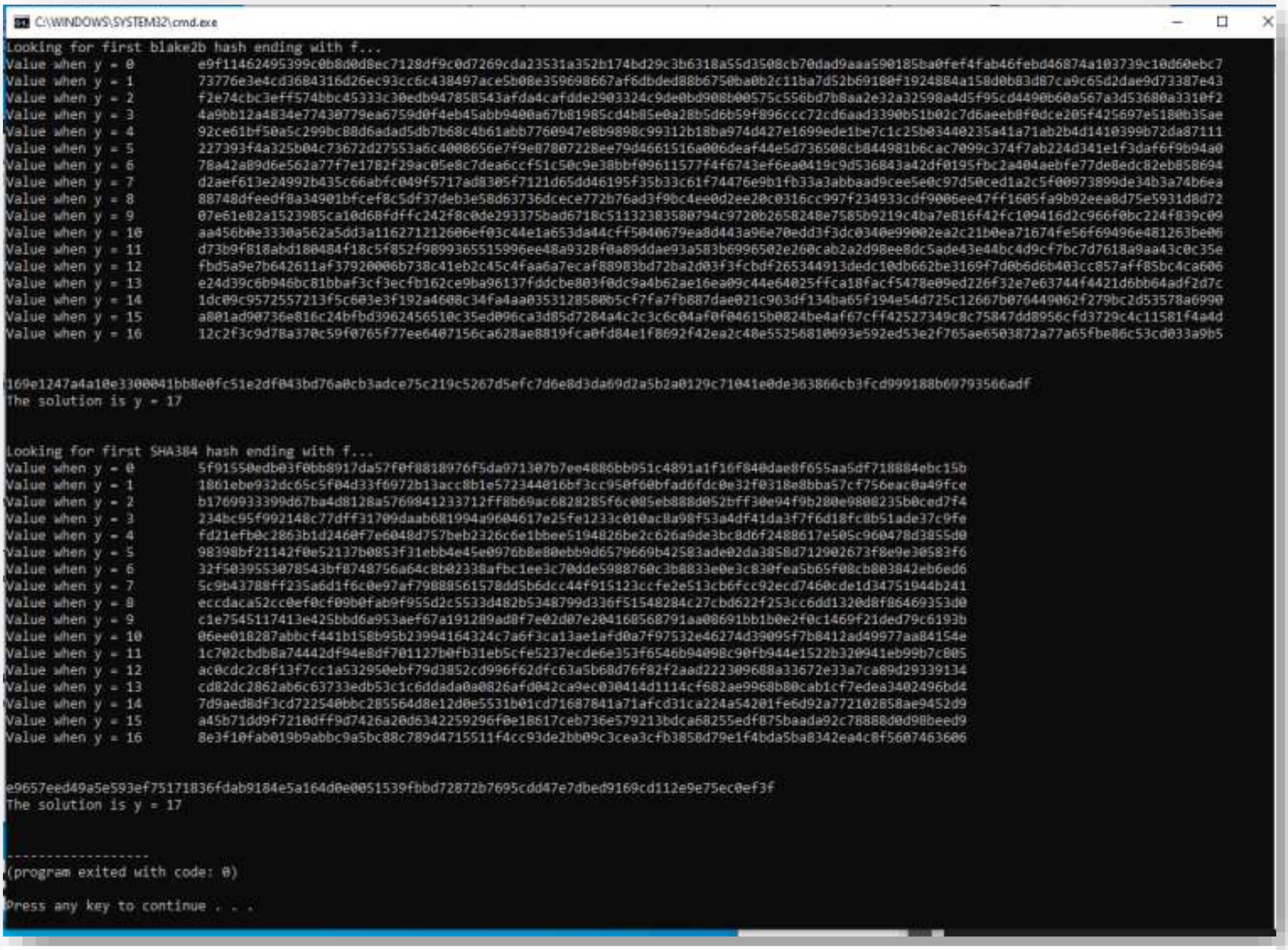
Proof of Work tests

I tested out the SHA384 and BLAKE2B hashing algorithms:



```python
"""
Proof of work (PoW) is a form of cryptographic zero-knowledge proof in
which one party (the prover) proves to others (the verifiers) that a
certain amount of computational effort has been
expended for some purpose. -https://en.wikipedia.org/wiki/Proof_of_work

We will use this PoW algorithm as part of creating new blocks on the
blockchain
"""

from hashlib import sha384
from hashlib import blake2b

x, y = 5, 0

"""Here we will brute-force what number y needs to equal for the
generated hash to end in f. The resulting hash value will always be
the same using this number for y.
"""

#Test with BLAKE2B
print("Looking for first blake2b hash ending with f...")
while blake2b(f'{x*y}'.encode()).hexdigest()[-1] != "f":
    print(f"Value when y = {y}\t"+ blake2b(f'{x*y}'.encode()).hexdigest())
    y += 1
    
print("\n\n"+ blake2b(f'{x*y}'.encode()).hexdigest())
print(f'The solution is y = {y}')

#Test with SHA384
x, y = 5, 0
print("\n\nLooking for first SHA384 hash ending with f...")
while sha384(f'{x*y}'.encode()).hexdigest()[-1] != "f":
    print(f"Value when y = {y}\t"+ sha384(f'{x*y}'.encode()).hexdigest())
    y += 1
    
print("\n\n"+ sha384(f'{x*y}'.encode()).hexdigest())
print(f'The solution is y = {y}')
```

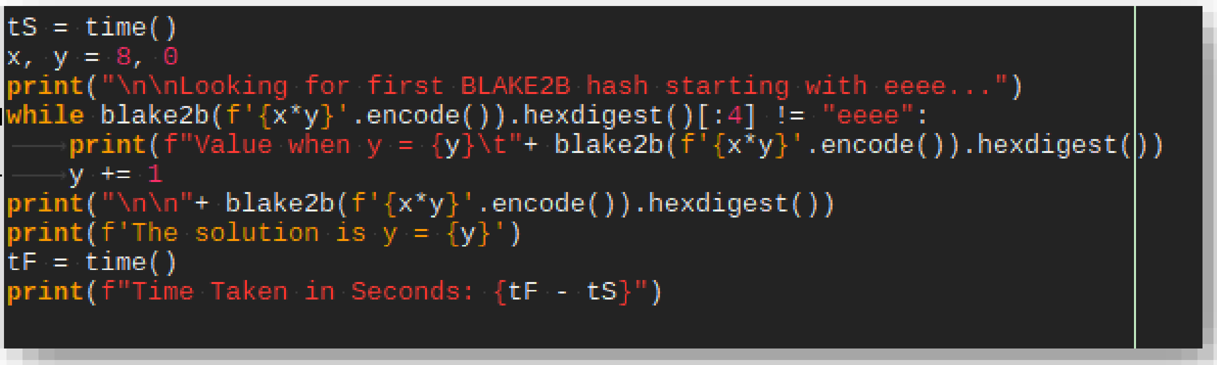
Figure 45: PoW test code



```
C:\WINDOWS\SYSTEM32\cmd.exe
Looking for first blake2b hash ending with f...
Value when y = 0        e9f11462495399c0b8d0d8ec7128df9c0d7269cda23531a352b174bd29c3b6318a55d3508cb70dad9aaa590185ba0fef4fab46febd46874a103739c10d60ebc7
Value when y = 1        73776e3e4cd3684316d26ec93cc6c438497ace5b08e359698667af6dbded88b6750ba0b2c11ba7d52b69180f1924884a158d0b83d87ca9c65d2dae9d73387e43
Value when y = 2        f2e74cbc3eff574bbc45333c30edb947858543afda4cafdde2903324c9de0bd908b00575c556bd7b8aa2e32a32598a4d5f95cd4490b60a567a3d53680a3310f2
Value when y = 3        4a9bb12a4834e77430779ea6759d0f4eb45abb9400a67b81985cd4b85e0a28b5d6b59f896ccc72cd6aad3390b51b02c7d6aeeb8f0dce205f425697e5180b35ae
Value when y = 4        92ce61bf50a5c299bc88d6adad5db7b68c4b61abb7760947e8b9898c99312b18ba974d427e1699ede1be7c1c25b03440235a41a71ab2b4d1410399b72da87111
Value when y = 5        227393f4a325b04c73672d27553a6c4008656e7f9e87807228ee79d4661516a006deaf44e5d736508cb844981b6cac7099c374f7ab224d341e1f3daf6f9b94a0
Value when y = 6        78a42a89d6e562a77f7e1782f29ac05e8c7dea6ccf51c50c9e38bbf09611577f4f6743ef6ea0419c9d536843a42df0195fbc2a404aebfe77de8edc82eb858694
Value when y = 7        d2aef613e24992b435c66abfc049f5717ad8305f7121d65dd46195f35b33c61f74476e9b1fb33a3abbaad9cee5e0c97d50ced1a2c5f00973899de34b3a74b6ea
Value when y = 8        88748dfeedf8a34901bfcef8c5df37deb3e58d63736dcece772b76ad3f9bc4ee0d2ee20c0316cc997f234933cdf9006ee47ff1605fa9b92eea8d75e5931d8d72
Value when y = 9        07e61e82a1523985ca10d68fdffc242f8c0de293375bad6718c51132383580794c9720b2658248e7585b9219c4ba7e816f42fc109416d2c966f0bc224f839c09
Value when y = 10       aa456b0e3330a562a5dd3a116271212606ef03c44e1a653da44cff5040679ea8d443a96e70edd3f3dc0340e99002ea2c21b0ea71674fe56f69496e481263be06
Value when y = 11       d73b9f818abd180484f18c5f852f9899365515996ee48a9328f0a89ddae93a583b6996502e260cab2a2d98ee8dc5ade43e44bc4d9cf7bc7d7618a9aa43c0c35e
Value when y = 12       fbd5a9e7b642611af37920006b738c41eb2c45c4faa6a7ecaf88983bd72ba2d03f3fcbdf265344913dedc10db662be3169f7d0b6d6b403cc857aff85bc4ca606
Value when y = 13       e24d39c6b946bc81bbaf3cf3ecfb162ce9ba96137fddcbe803f0dc9a4b62ae16ea09c44e64025ffca18facf5478e09ed226f32e7e63744f4421d6bb64adf2d7c
Value when y = 14       1dc09c9572557213f5c603e3f192a4608c34fa4aa0353128580b5cf7fa7fb887dae021c963df134ba65f194e54d725c12667b076449062f279bc2d53578a6990
Value when y = 15       a801ad90736e816c24bfbd3962456510c35ed096ca3d85d7284a4c2c3c6c04af0f04615b0824be4af67cff42527349c8c75847dd8956cfd3729c4c11581f4a4d
Value when y = 16       12c2f3c9d78a370c59f0765f77ee6407156ca628ae8819fca0fd84e1f8692f42ea2c48e55256810693e592ed53e2f765ae6503872a77a65fbe86c53cd033a9b5


169e1247a4a10e3300041bb8e0fc51e2df043bd76a0cb3adce75c219c5267d5efc7d6e8d3da69d2a5b2a0129c71041e0de363866cb3fcd999188b69793566adf
The solution is y = 17


Looking for first SHA384 hash ending with f...
Value when y = 0        5f91550edb03f0bb8917da57f0f8818976f5da971307b7ee4886bb951c4891a1f16f840dae8f655aa5df718884ebc15b
Value when y = 1        1861ebe932dc65c5f04d33f6972b13acc8b1e572344016bf3cc950f60bfad6fdc0e32f0318e8bba57cf756eac0a49fce
Value when y = 2        b1769933399d67ba4d8128a5769841233712ff8b69ac6828285f6c085eb888d052bff30e94f9b280e9800235b0ced7f4
Value when y = 3        234bc95f992148c77dff31709daab681994a9604617e25fe1233c010ac8a98f53a4df41da3f7f6d18fc8b51ade37c9fe
Value when y = 4        fd21efb0c2863b1d2460f7e6048d757beb2326c6e1bbee5194826be2c626a9de3bc8d6f2488617e505c960478d3855d0
Value when y = 5        98398bf21142f0e52137b0853f31ebb4e45e0976b8e80ebb9d6579669b42583ade02da3858d712902673f8e9e30583f6
Value when y = 6        32f5039553078543bf8748756a64c8b02338afbc1ee3c70dde5988760c3b8833e0e3c830fea5b65f08cb803842eb6ed6
Value when y = 7        5c9b43788ff235a6d1f6c0e97af79888561578dd5b6dcc44f915123ccfe2e513cb6fcc92ecd7460cde1d34751944b241
Value when y = 8        eccdaca52cc0ef0cf09b0fab9f955d2c5533d482b5348799d336f51548284c27cbd622f253cc6dd1320d8f86469353d0
Value when y = 9        c1e7545117413e425bbd6a953aef67a191289ad8f7e02d07e204168568791aa08691bb1b0e2f0c1469f21ded79c6193b
Value when y = 10       06ee018287abbcf441b158b95b23994164324c7a6f3ca13ae1afd0a7f97532e46274d39095f7b8412ad49977aa84154e
Value when y = 11       1c702cbdb8a74442df94e8df701127b0fb31eb5cfe5237ecde6e353f6546b94098c90fb944e1522b320941eb99b7c805
Value when y = 12       ac0cdc2c8f13f7cc1a532950ebf79d3852cd996f62dfc63a5b68d76f82f2aad222309688a33672e33a7ca89d29339134
Value when y = 13       cd82dc2862ab6c63733edb53c1c6ddada0a0826afd042ca9ec030414d1114cf682ae9968b80cab1cf7edea3402496bd4
Value when y = 14       7d9aed8df3cd722540bbc285564d8e12d0e5531b01cd71687841a71afcd31ca224a54201fe6d92a772102858ae9452d9
Value when y = 15       a45b71dd9f7210dff9d7426a20d6342259296f0e18617ceb736e579213bdca68255edf875baada92c78888d0d98beed9
Value when y = 16       8e3f10fab019b9abbc9a5bc88c789d4715511f4cc93de2bb09c3cea3cfb3858d79e1f4bda5ba8342ea4c8f5607463606


e9657eed49a5e593ef75171836fdab9184e5a164d0e0051539fbbd72872b7695cdd47e7dbed9169cd112e9e75ec0ef3f
The solution is y = 17



------------------
(program exited with code: 0)

Press any key to continue . . .
```

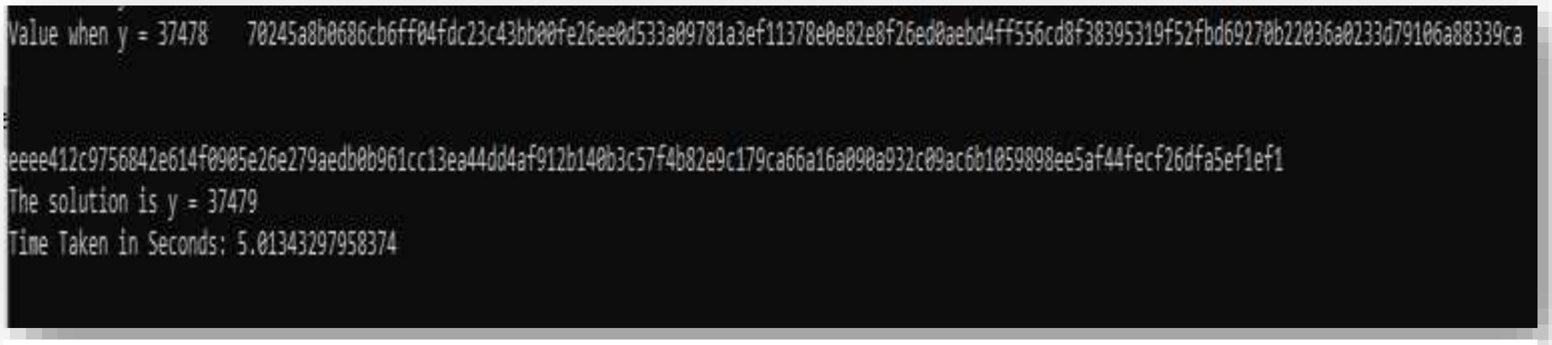
Figure 46: PoW test result

Then I created a simple test with BLAKE2B to find the first hash value that contains four leading e characters:



```python
tS = time()
x, y = 8, 0
print("\n\nLooking for first BLAKE2B hash starting with eeee...")
while blake2b(f'{x*y}'.encode()).hexdigest()[:4] != "eeee":
    print(f"Value when y = {y}\t"+ blake2b(f'{x*y}'.encode()).hexdigest())
    y += 1
print("\n\n"+ blake2b(f'{x*y}'.encode()).hexdigest())
print(f'The solution is y = {y}')
tF = time()
print(f"Time Taken in Seconds: {tF - tS}")
```

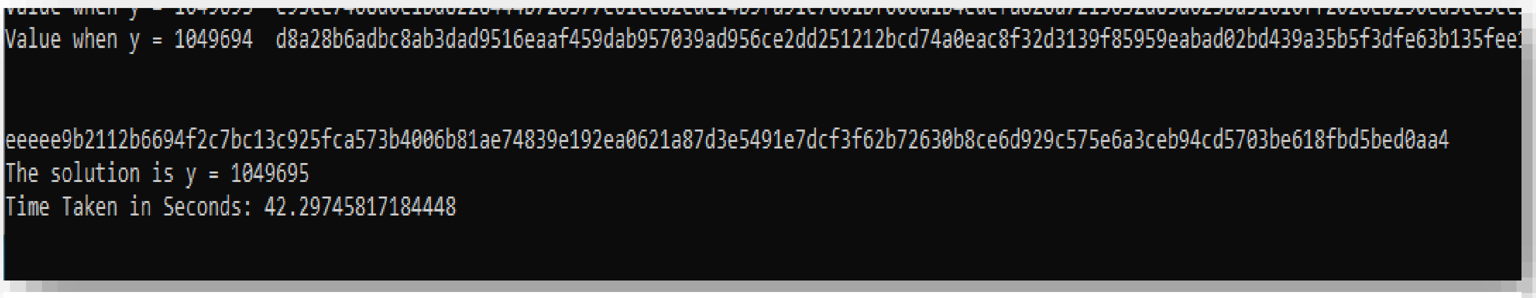
Figure 47: PoW test 2 code



```
Value when y = 37478   70245a8b0686cb6ff04fdc23c43bb00fe26ee0d533a09781a3ef11378e0e82e8f26ed0aebd4ff556cd8f38395319f52fbd69270b22036a0233d79106a88339ca


eeee412c9756842e614f0905e26e279aedb0b961cc13ea44dd4af912b140b3c57f4b82e9c179ca66a16a090a932c09ac6b1059898ee5af44fecf26dfa5ef1ef1
The solution is y = 37479
Time Taken in Seconds: 5.01343297958374
```

Figure 49: PoW test 2 result 1

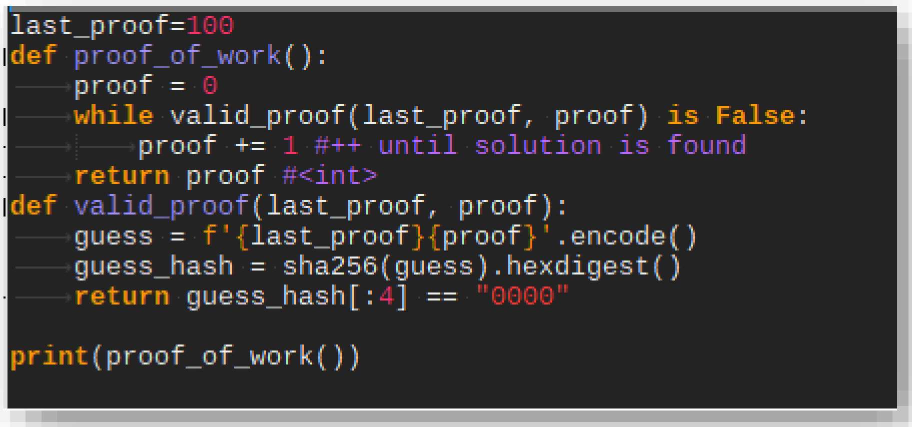
I increased this to five leading e's, notice the time difference:



```
Value when y = 1049694  d8a28b6adbc8ab3dad9516eaaf459dab957039ad956ce2dd251212bcd74a0eac8f32d3139f85959eabad02bd439a35b5f3dfe63b135fee


eeeee9b2112b6694f2c7bc13c925fca573b4006b81ae74839e192ea0621a87d3e5491e7dcf3f62b72630b8ce6d929c575e6a3ceb94cd5703be618fbd5bed0aa4
The solution is y = 1049695
Time Taken in Seconds: 42.29745817184448
```

Figure 48: PoW test 2 result 2

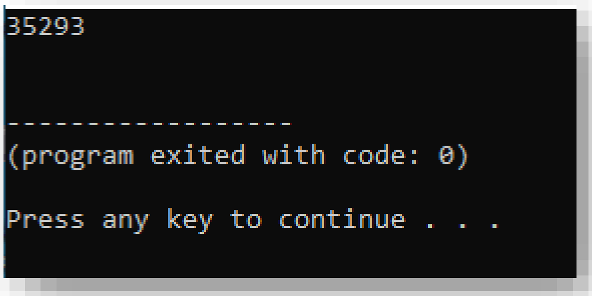
Finally I created a mini simulation of what we are using in the program itself



```
last_proof=100
def proof_of_work():
    proof = 0
    while valid_proof(last_proof, proof) is False:
        proof += 1 #++ until solution is found
    return proof #<int>
def valid_proof(last_proof, proof):
    guess = f'{last_proof}{proof}'.encode()
    guess_hash = sha256(guess).hexdigest()
    return guess_hash[:4] == "0000"

print(proof_of_work())
```

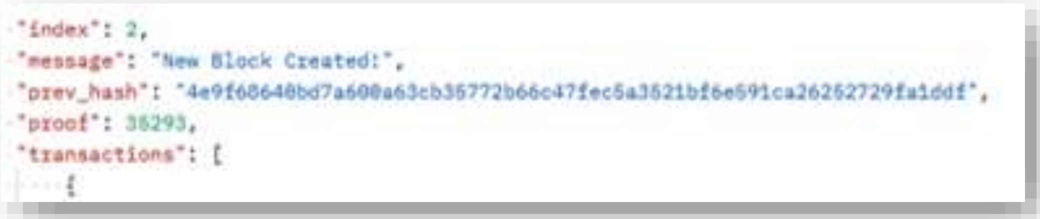
Figure 50: PoW test 3

We get the same result as the proof value we got for block 2 in our initial block chain test: 35293



```
35293


------------------
(program exited with code: 0)

Press any key to continue . . .
```

Figure 51: PoW test 3 result

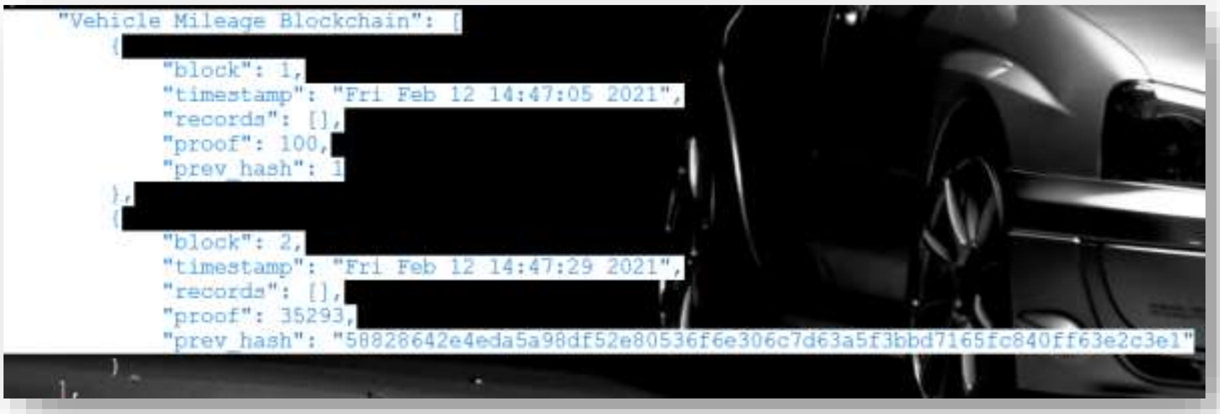


```
"index": 2,
"message": "New Block Created:",
"prev_hash": "4e9f68640bd7a600a63cb35772b66c47fec5a3521bf6e591ca26262729fa1ddf",
"proof": 35293,
"transactions": [
    {
```

Figure 52: PoW test 3 proof compare

Hashing test: Here I did a simple run-through of the program to get a generated hash value and try to recreate it using the same information.

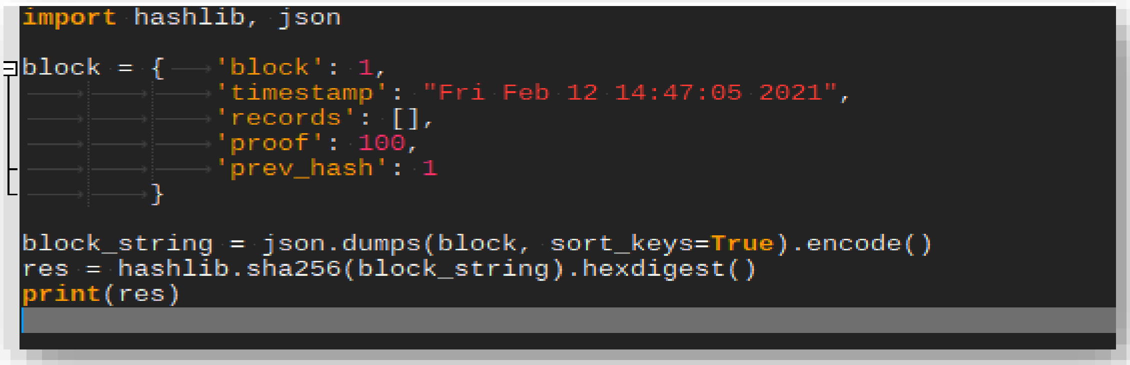
The SHA256 hash value I got: 58828642e4eda5a98df52e80536f6e306c7d63a5f3bbd7165fc840ff63e2c3e1



```
"Vehicle Mileage Blockchain": [
    {
        "block": 1,
        "timestamp": "Fri Feb 12 14:47:05 2021",
        "records": [],
        "proof": 100,
        "prev_hash": 1
    },
    {
        "block": 2,
        "timestamp": "Fri Feb 12 14:47:29 2021",
        "records": [],
        "proof": 35293,
        "prev_hash": "58828642e4eda5a98df52e80536f6e306c7d63a5f3bbd7165fc840ff63e2c3e1"
    }
],
```

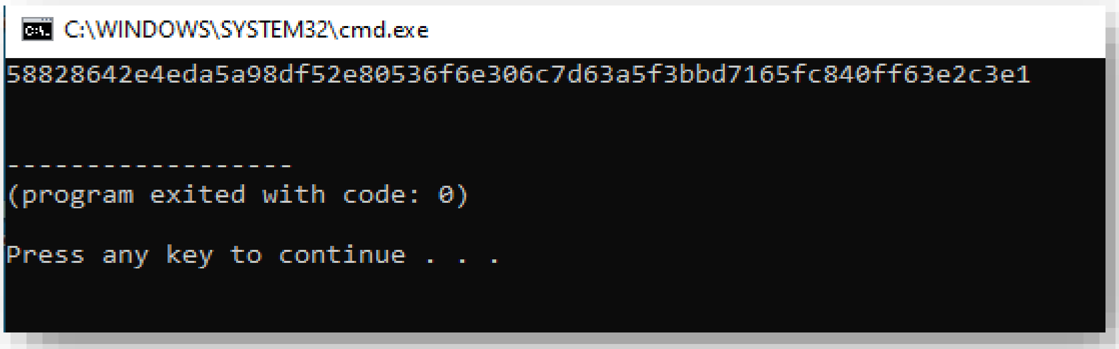
Figure 53: hash test original

I re-ran this info through the same SHA256 algorithm to produce the same result:



```
import hashlib, json

block = {    'block': 1,
             'timestamp': "Fri Feb 12 14:47:05 2021",
             'records': [],
             'proof': 100,
             'prev_hash': 1
        }

block_string = json.dumps(block, sort_keys=True).encode()
res = hashlib.sha256(block_string).hexdigest()
print(res)
```

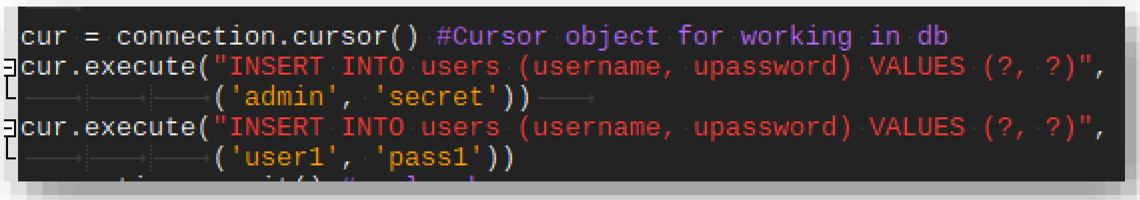
Figure 54: hash test code



```
C:\WINDOWS\SYSTEM32\cmd.exe
58828642e4eda5a98df52e80536f6e306c7d63a5f3bbd7165fc840ff63e2c3e1


------------------
(program exited with code: 0)

Press any key to continue . . .
```

Figure 55: hash test result

Login Test: In order to add new blocks to the chain a user must first login, I created a simple database for this purpose and did some testing. First our database users:



```
cur = connection.cursor() #Cursor object for working in db
cur.execute("INSERT INTO users (username, upassword) VALUES (?, ?)",
            ('admin', 'secret'))
cur.execute("INSERT INTO users (username, upassword) VALUES (?, ?)",
            ('user1', 'pass1'))
```

Figure 56: login test users

Our login page:

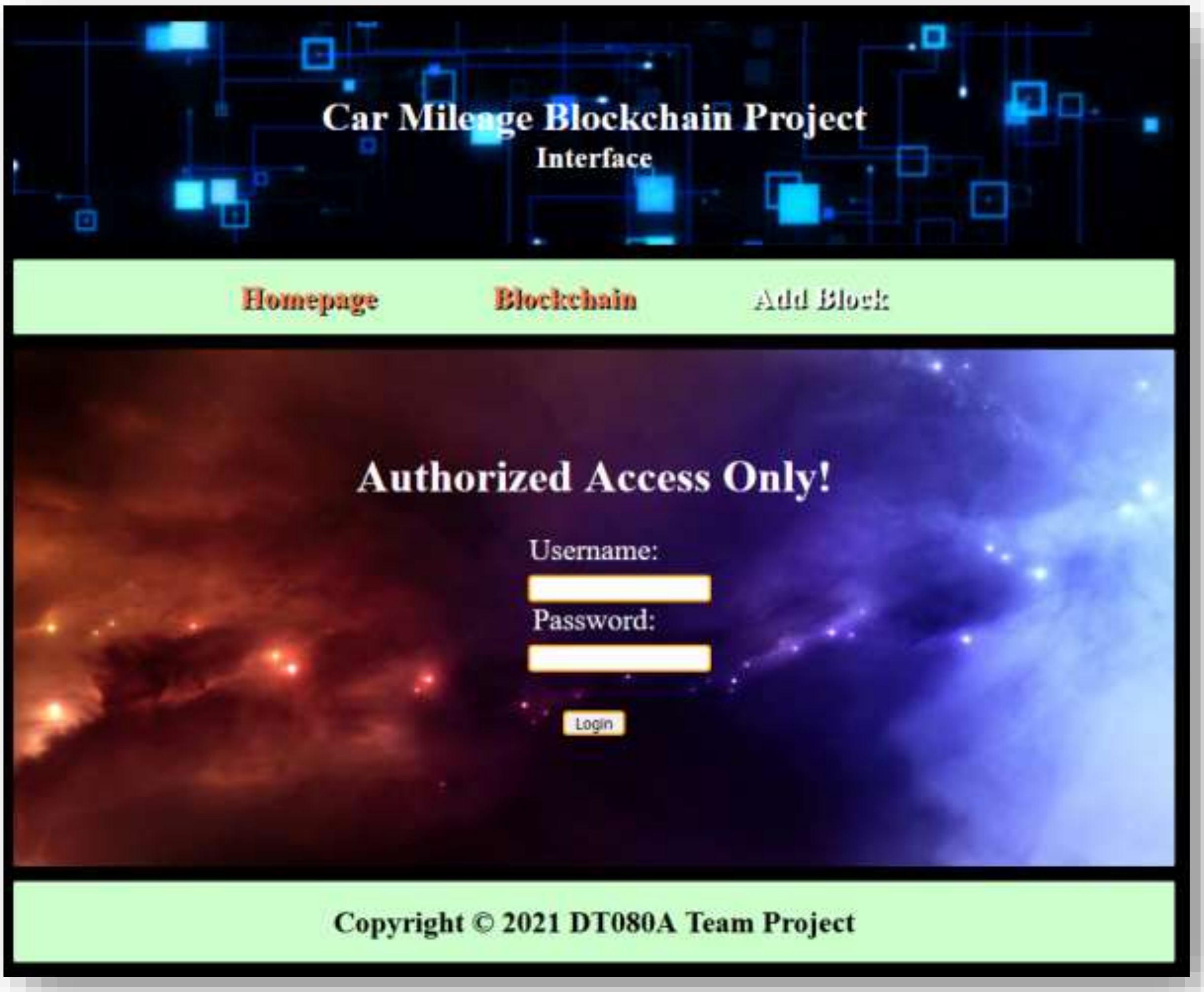
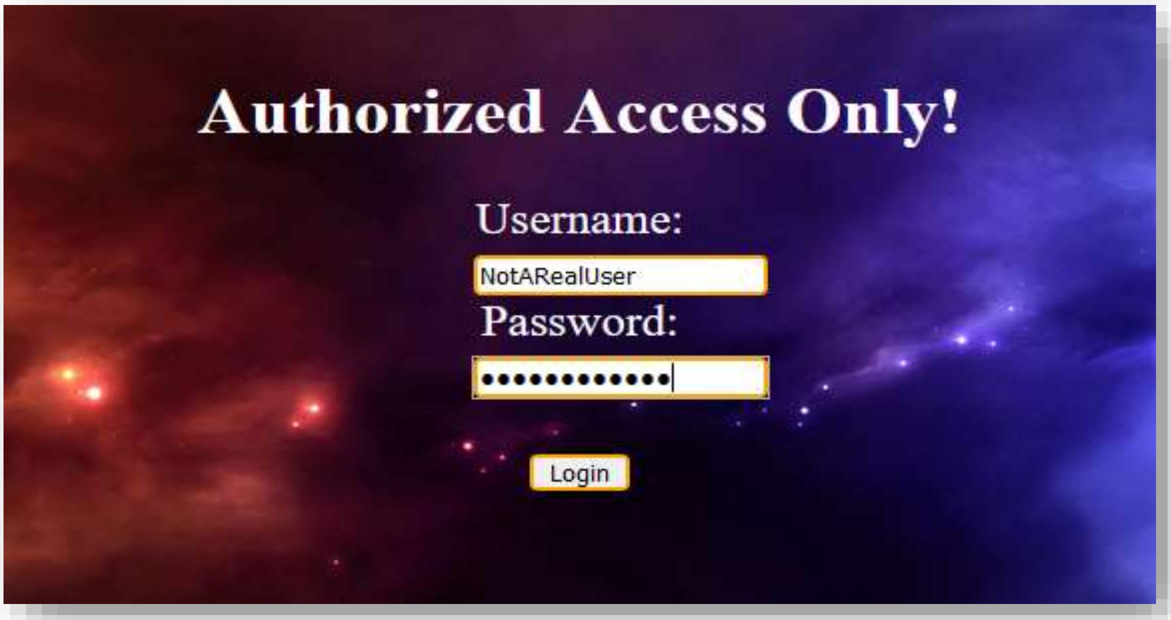


Figure 57: login form

If we try and login a non-existing user we get an error message:



Authorized Access Only!

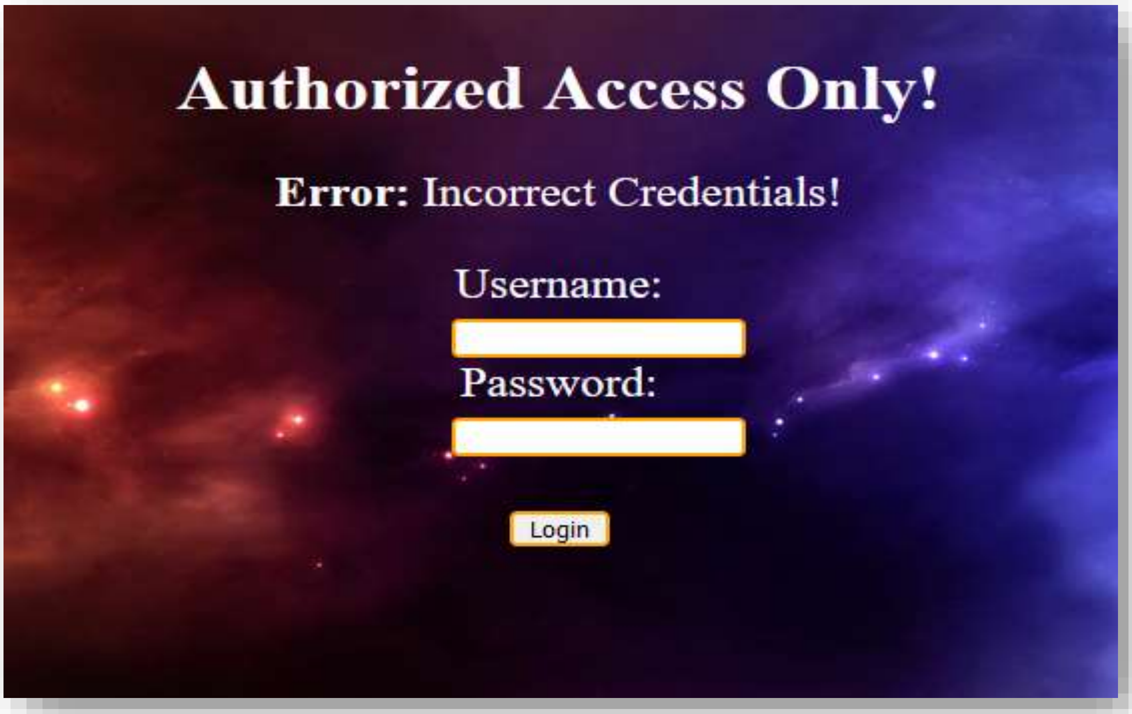
Username:

NotARealUser

Password:

•••••••••••••

Figure 58: non-user login attempt



Authorized Access Only!

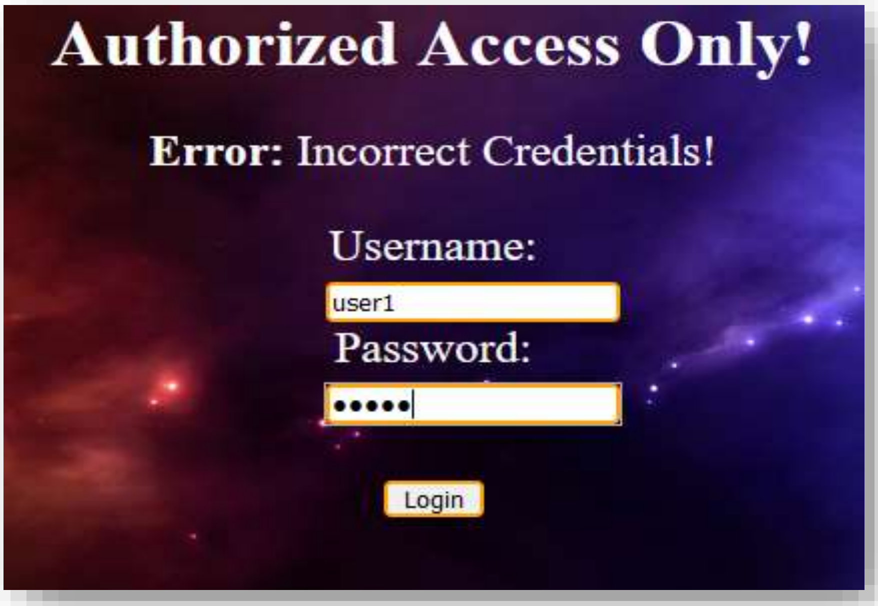
Error: Incorrect Credentials!

Username:

Password:

Figure 59: non-user login result

We repeat this process for a valid user account:


Authorized Access Only!

Error: Incorrect Credentials!

Username:

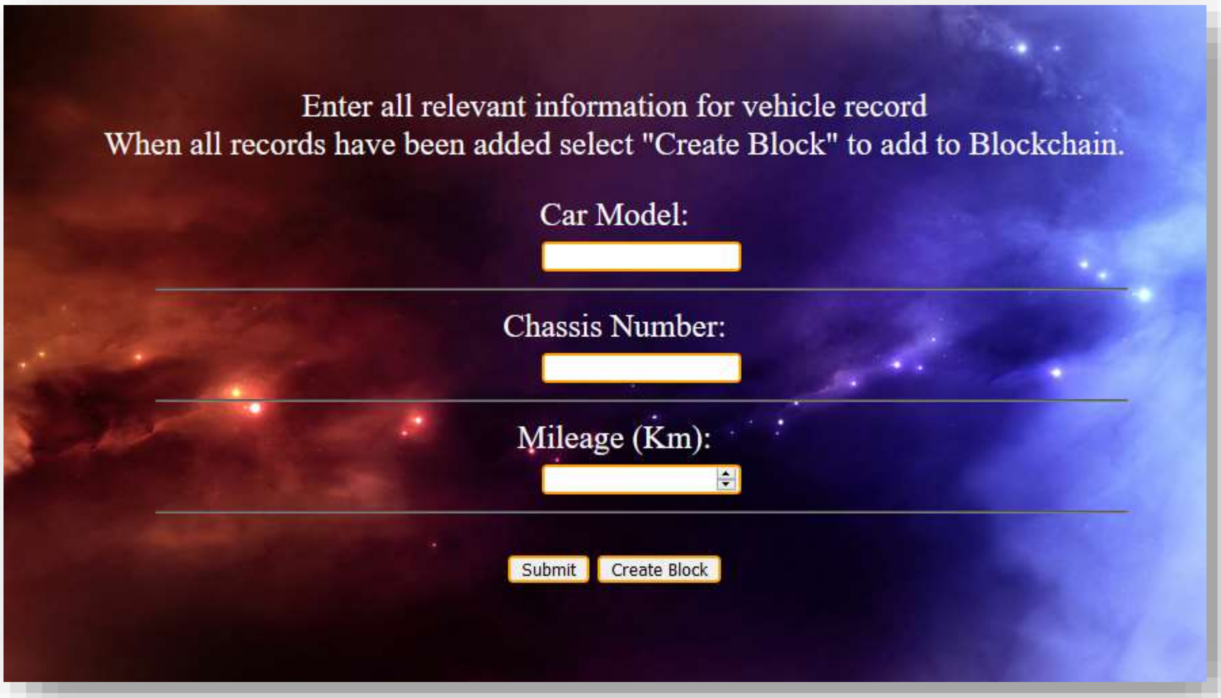
user1

Password:

•••••

Figure 60: Valid user login attempt

We then get access to the add record page:


Enter all relevant information for vehicle record
When all records have been added select "Create Block" to add to Blockchain.

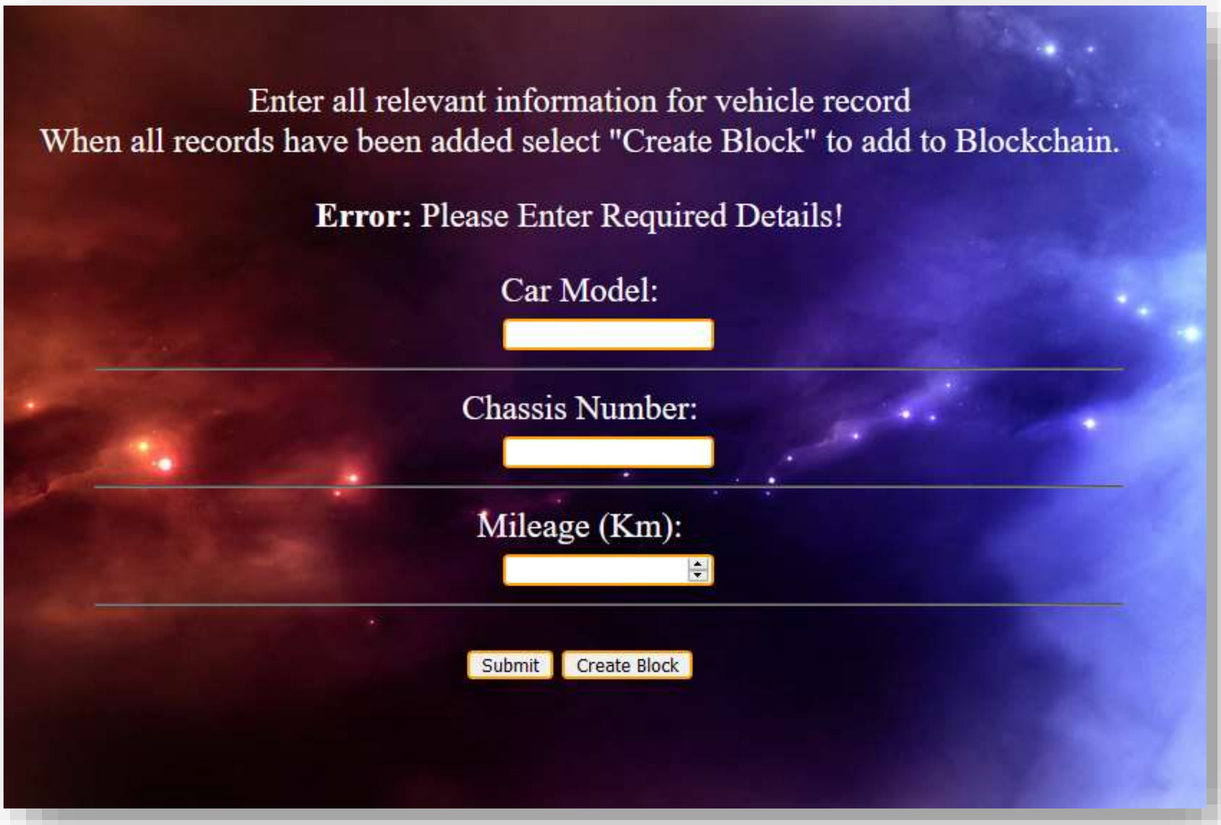
Car Model:

Chassis Number:

Mileage (Km):

Figure 61: valid user login result

Again if any of the required fields here are missing an error message is displayed on screen:



Enter all relevant information for vehicle record
When all records have been added select "Create Block" to add to Blockchain.

Error: Please Enter Required Details!

Car Model:

Chassis Number:

Mileage (Km):

Figure 62: adding empty record result

Summary

Overall I found this project to be quite rewarding, I learned a lot about Blockchain infrastructures and utilizing Python with the Flask framework. I am disappointed that we couldn’t fully implement the P2P node network functionality, I had a prototype version in which I could add nodes via a LAN to the overall infrastructure, but extending this functionality any further would have proven too time consuming.

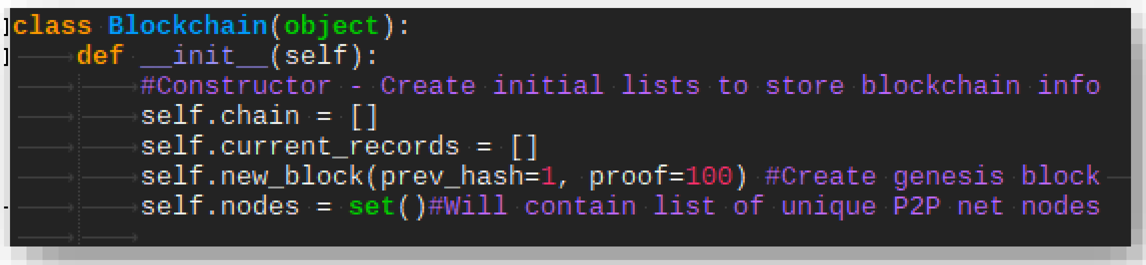
The blockchain framework built for this project provides the building blocks for any type of blockchain application, a vehicle mileage ledger was chosen as a usage example, but the framework itself is not limited to just this. Using Python Objects and Classes to create the program provides a modular structure that can easily be modified for any specific use case.

The web application created in tandem with the blockchain provides a reliable platform in which case users can view the records and authorized entities can add records to the chain. The blocks are timestamped meaning a user could easily track the relevant info for whatever model or chassis number they supply.

Future of Project

Decentralized Nodes (P2P network): I had originally planned to setup this blockchain infrastructure based on a P2P network of nodes that would participate in the overall operation of it. I got a prototype version of this working within a LAN where each instance of the program would be assigned a unique ID and had the functionality to discover other nodes running the same program. I used a consensus algorithm that when run on a node, it would query all other nodes and if any of those nodes contained a blockchain longer than its own, it would replace its own blockchain with the longer one. This functionality had to be removed due to time and resource restraints.

Originally I had a self.nodes set attribute that each instance would have to store additional nodes:

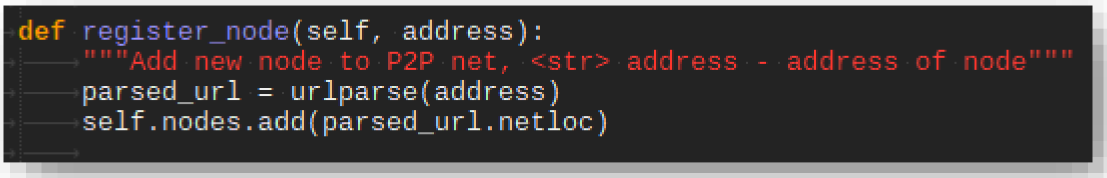


```
class Blockchain(object):
    def __init__(self):
        #Constructor - Create initial lists to store blockchain info
        self.chain = []
        self.current_records = []
        self.new_block(prev_hash=1, proof=100) #Create genesis block
        self.nodes = set()#Will contain list of unique P2P net nodes
```

Figure 63: constructor with nodes set

I had three Blockchain Class methods for node functionality: register_node(), valid_chain(), and resolve_conflicts().

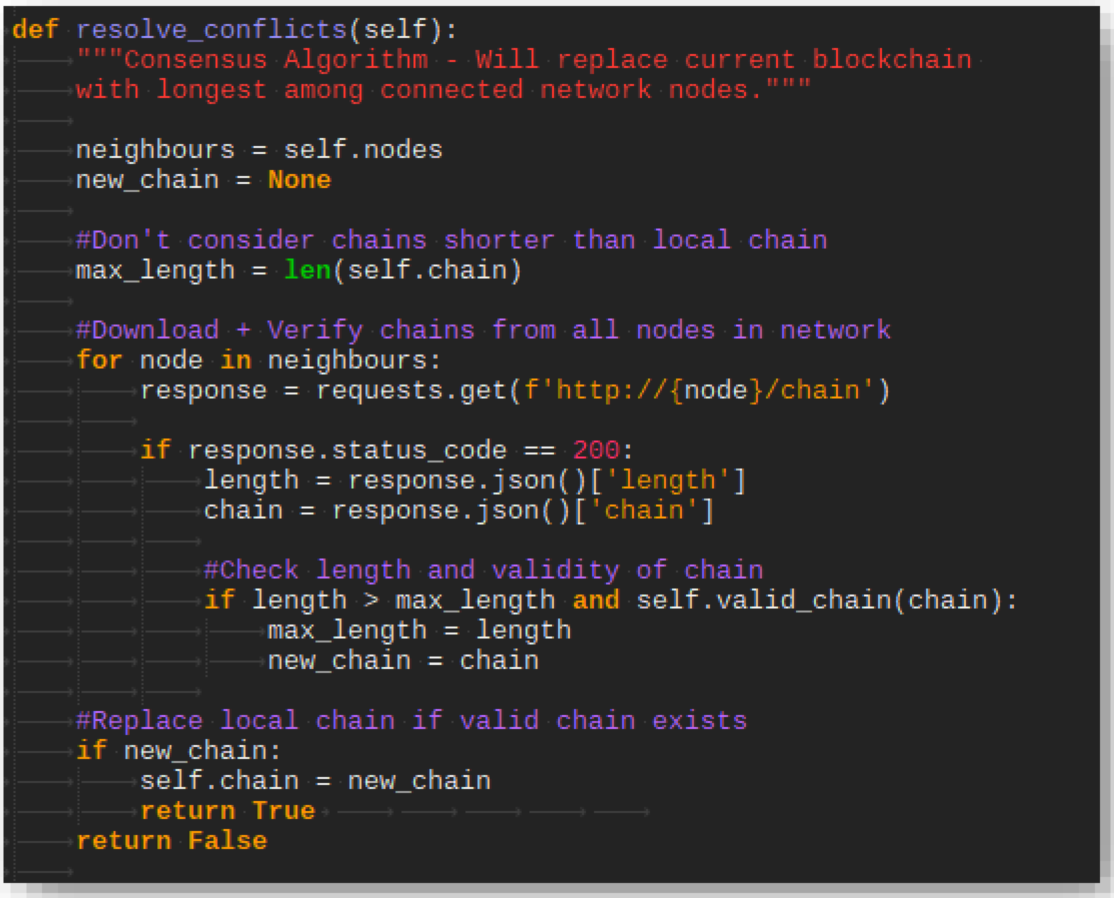
Register_node() simply took in a supplied network address (e.g. IPv4) and added it's network location to the previously created self.nodes set.



```
def register_node(self, address):
    """Add new node to P2P net, <str> address - address of node"""
    parsed_url = urlparse(address)
    self.nodes.add(parsed_url.netloc)
```

Figure 64: register_node()

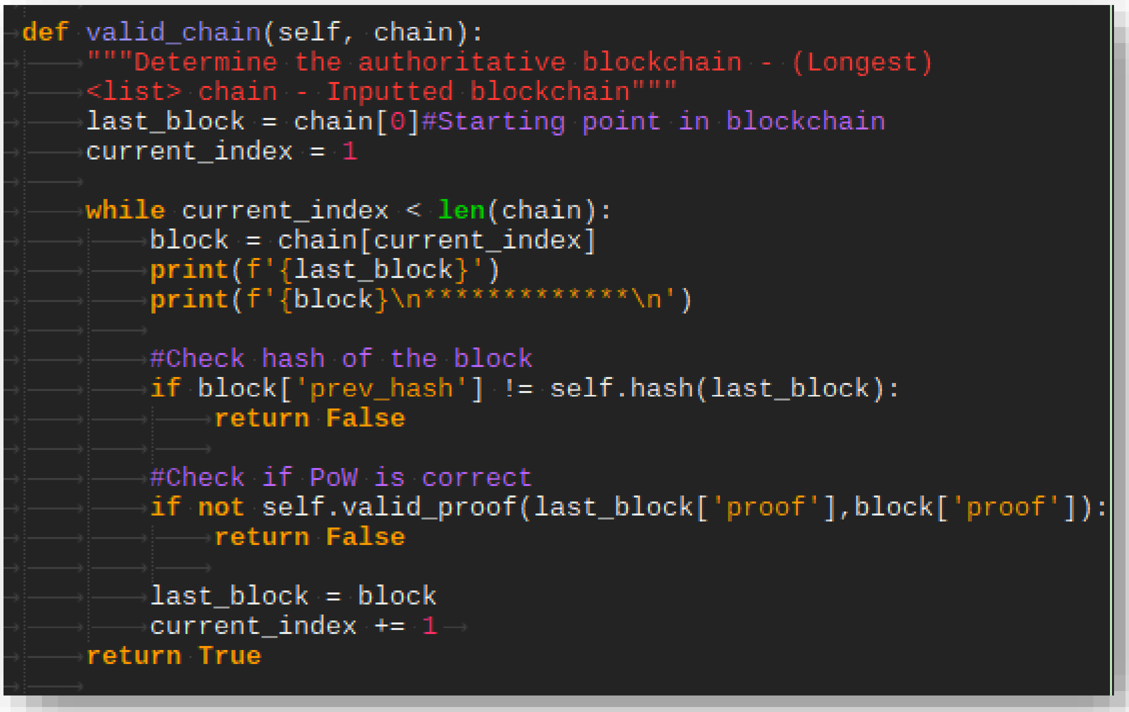
Resolve_conflicts() would act as the consensus algorithm, basically we would get all the nodes contained within self.nodes, iterate through them requesting their blockchains (discounting any shorter than the node's own chain), check the hashes and PoW is valid with valid_chain() and then replace the local chain with new one if a valid longer remote chain exists.



```python
def resolve_conflicts(self):
    """Consensus Algorithm - Will replace current blockchain
    with longest among connected network nodes."""

    neighbours = self.nodes
    new_chain = None

    #Don't consider chains shorter than local chain
    max_length = len(self.chain)

    #Download + Verify chains from all nodes in network
    for node in neighbours:
        response = requests.get(f'http://{node}/chain')

        if response.status_code == 200:
            length = response.json()['length']
            chain = response.json()['chain']

            #Check length and validity of chain
            if length > max_length and self.valid_chain(chain):
                max_length = length
                new_chain = chain

    #Replace local chain if valid chain exists
    if new_chain:
        self.chain = new_chain
        return True
    return False
```

Figure 65: resolve_conflicts()

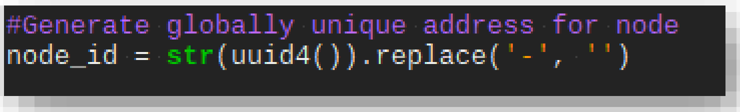
Valid_chain() would take in a supplied blockchain and iterate through each block, re-checking the hash and PoW for that Blockchain against its own to ensure the records contained within have not been tampered with.



```python
def valid_chain(self, chain):
    """Determine the authoritative blockchain - (Longest)
    <list> chain - Inputted blockchain"""
    last_block = chain[0]#Starting point in blockchain
    current_index = 1

    while current_index < len(chain):
        block = chain[current_index]
        print(f'{last_block}')
        print(f'{block}\n*************\n')

        #Check hash of the block
        if block['prev_hash'] != self.hash(last_block):
            return False

        #Check if PoW is correct
        if not self.valid_proof(last_block['proof'],block['proof']):
            return False

        last_block = block
        current_index += 1
    return True
```

Figure 66: valid_chain()

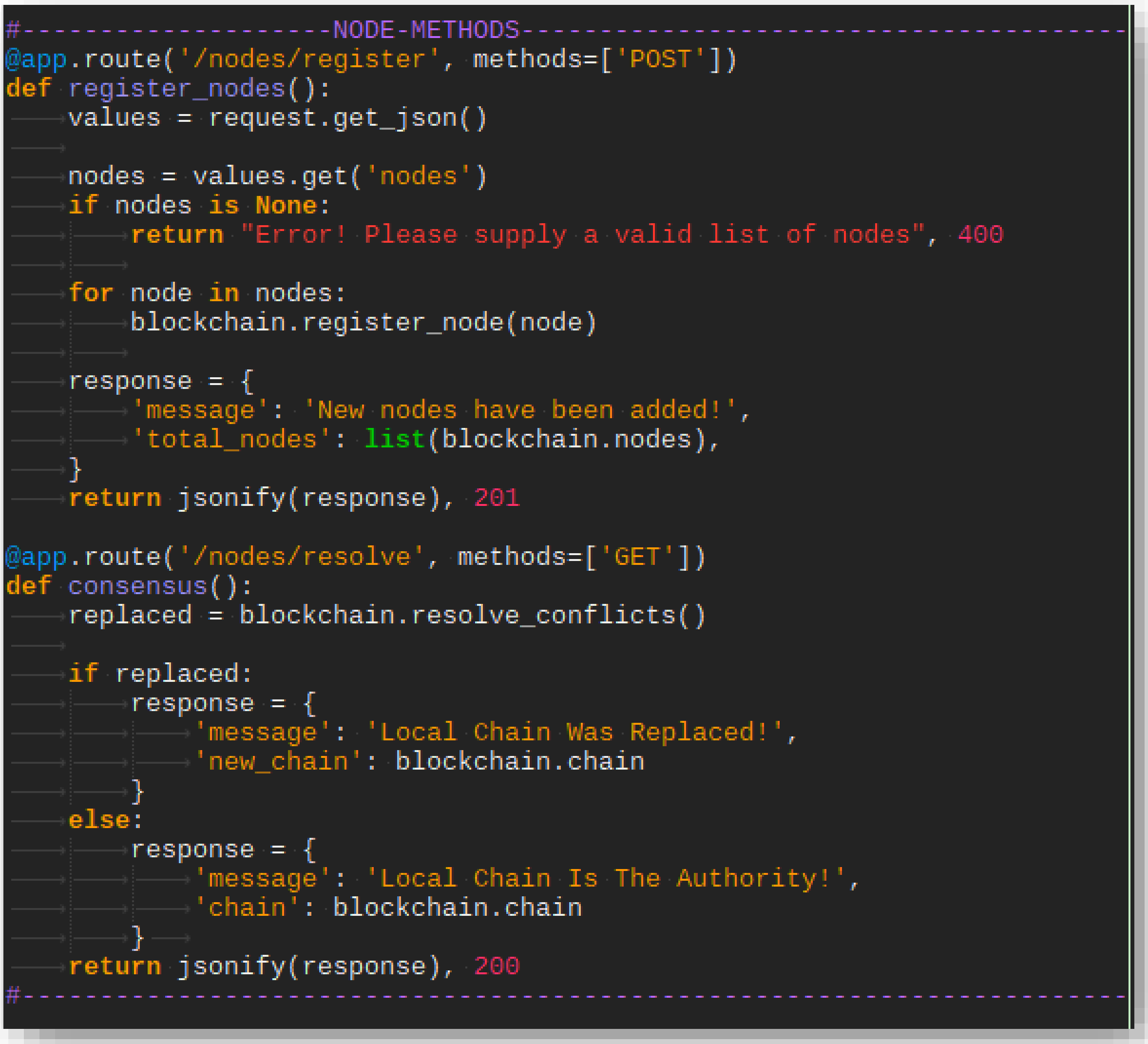
Additionally, at program start we would have to assign a unique ID to the node, the Python uuid4 module can be used to generate globally unique IDs:



```python
#Generate globally unique address for node
node_id = str(uuid4()).replace('-', '')
```

Figure 67: Creating unique node ID

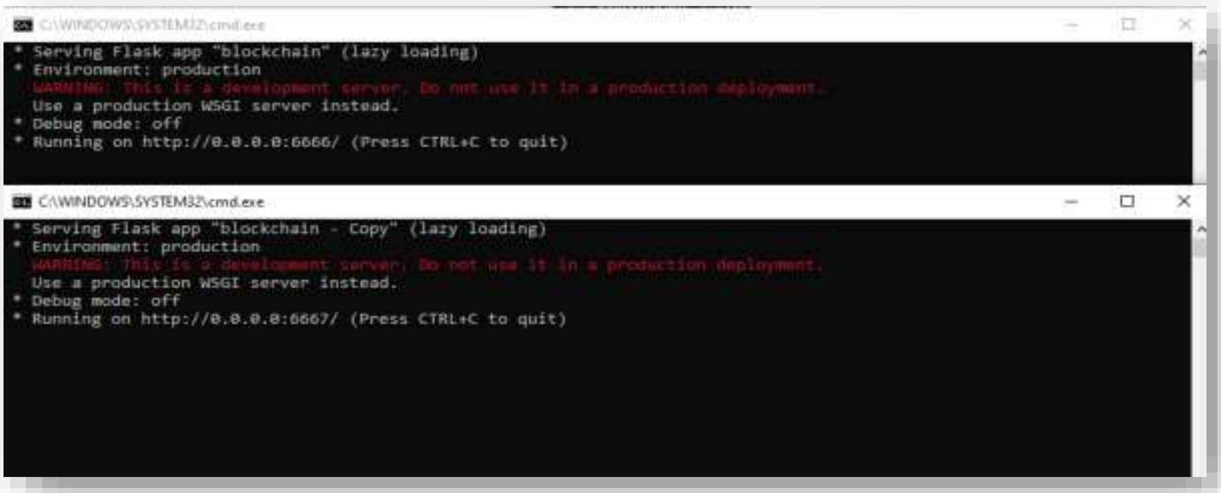
I also had global methods in which to interact with these class methods. For registering nodes we would get POSTed JSON data, add them to the node set, and return a response to the user. For resolving we would run the previously created resolve_conflicts() class method and inform the user whether the chain was replaced or not:



```
#--------------------NODE-METHODS--------------------------------------
@app.route('/nodes/register', methods=['POST'])
def register_nodes():
    values = request.get_json()

    nodes = values.get('nodes')
    if nodes is None:
        return "Error! Please supply a valid list of nodes", 400

    for node in nodes:
        blockchain.register_node(node)

    response = {
        'message': 'New nodes have been added!',
        'total_nodes': list(blockchain.nodes),
    }
    return jsonify(response), 201

@app.route('/nodes/resolve', methods=['GET'])
def consensus():
    replaced = blockchain.resolve_conflicts()

    if replaced:
        response = {
            'message': 'Local Chain Was Replaced!',
            'new_chain': blockchain.chain
        }
    else:
        response = {
            'message': 'Local Chain Is The Authority!',
            'chain': blockchain.chain
        }
    return jsonify(response), 200
#----------------------------------------------------------------------
```

Figure 68: node global methods

I performed a simple test, first starting two different nodes:

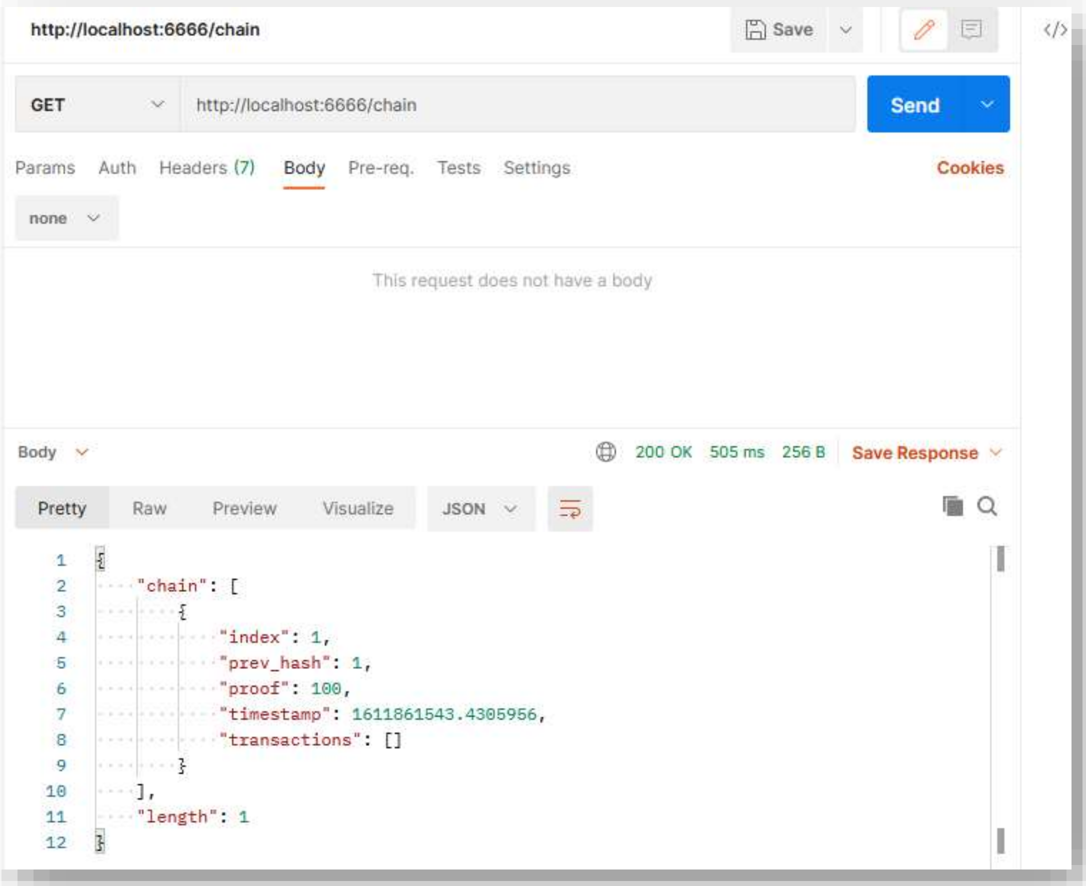


```
 * Serving Flask app "blockchain" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:6666/ (Press CTRL+C to quit)
```

```
 * Serving Flask app "blockchain - Copy" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:6667/ (Press CTRL+C to quit)
```

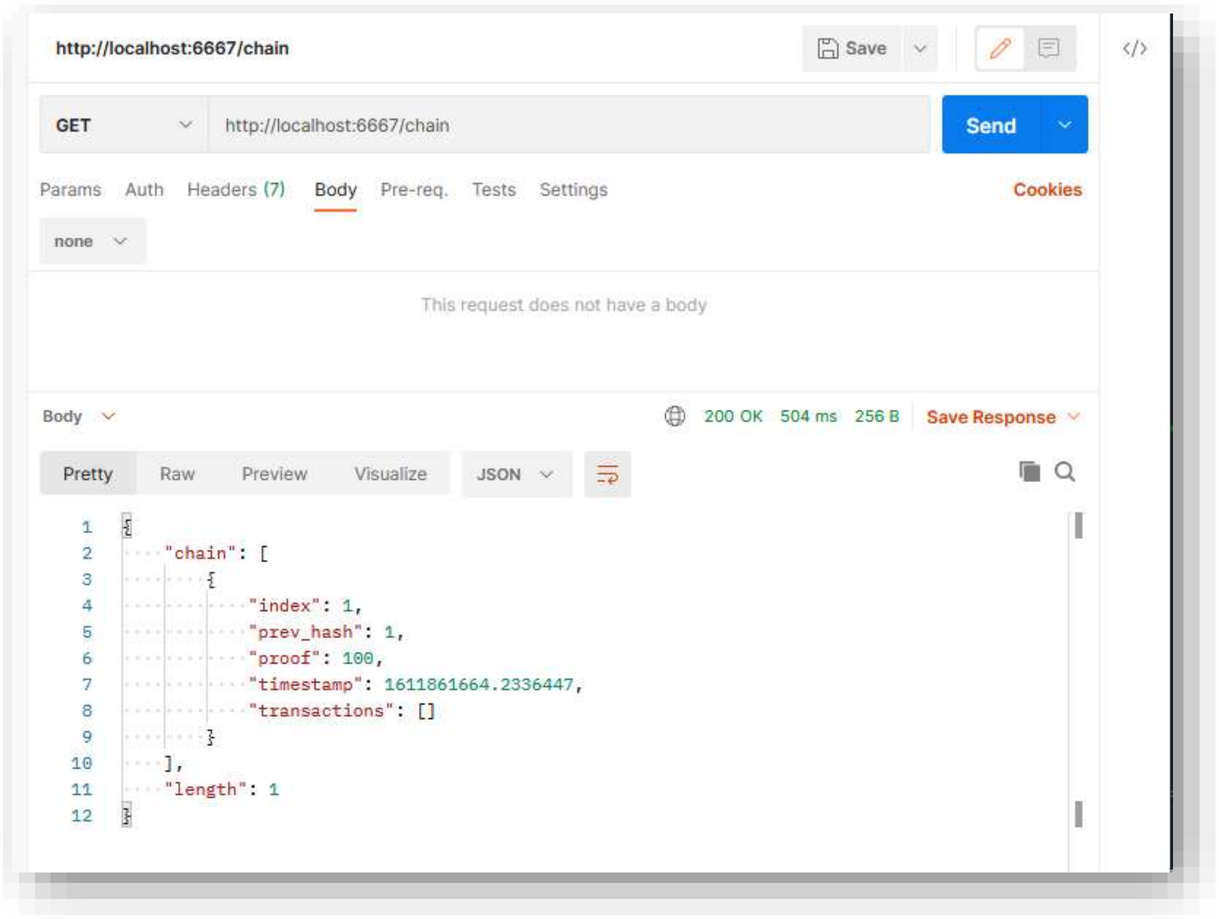
Figure 69: svr start

Then checking their initial chains:



```
{
    "chain": [
        {
            "index": 1,
            "prev_hash": 1,
            "proof": 100,
            "timestamp": 1611861543.4305956,
            "transactions": []
        }
    ],
    "length": 1
}
```

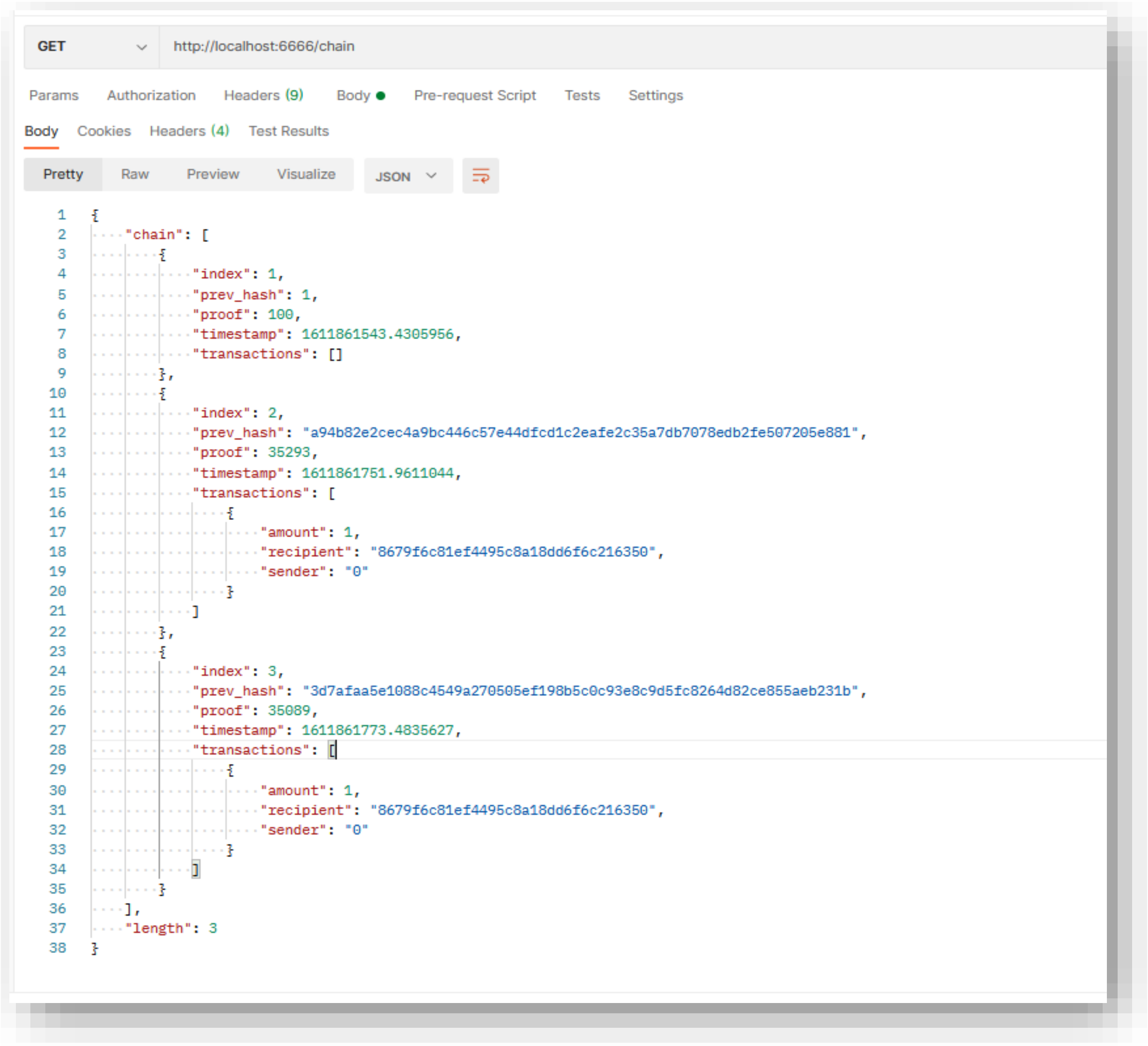
Figure 70: svr 6666 init chain



```
{
    "chain": [
        {
            "index": 1,
            "prev_hash": 1,
            "proof": 100,
            "timestamp": 1611861664.2336447,
            "transactions": []
        }
    ],
    "length": 1
}
```

Figure 71: svr 6667 init chain

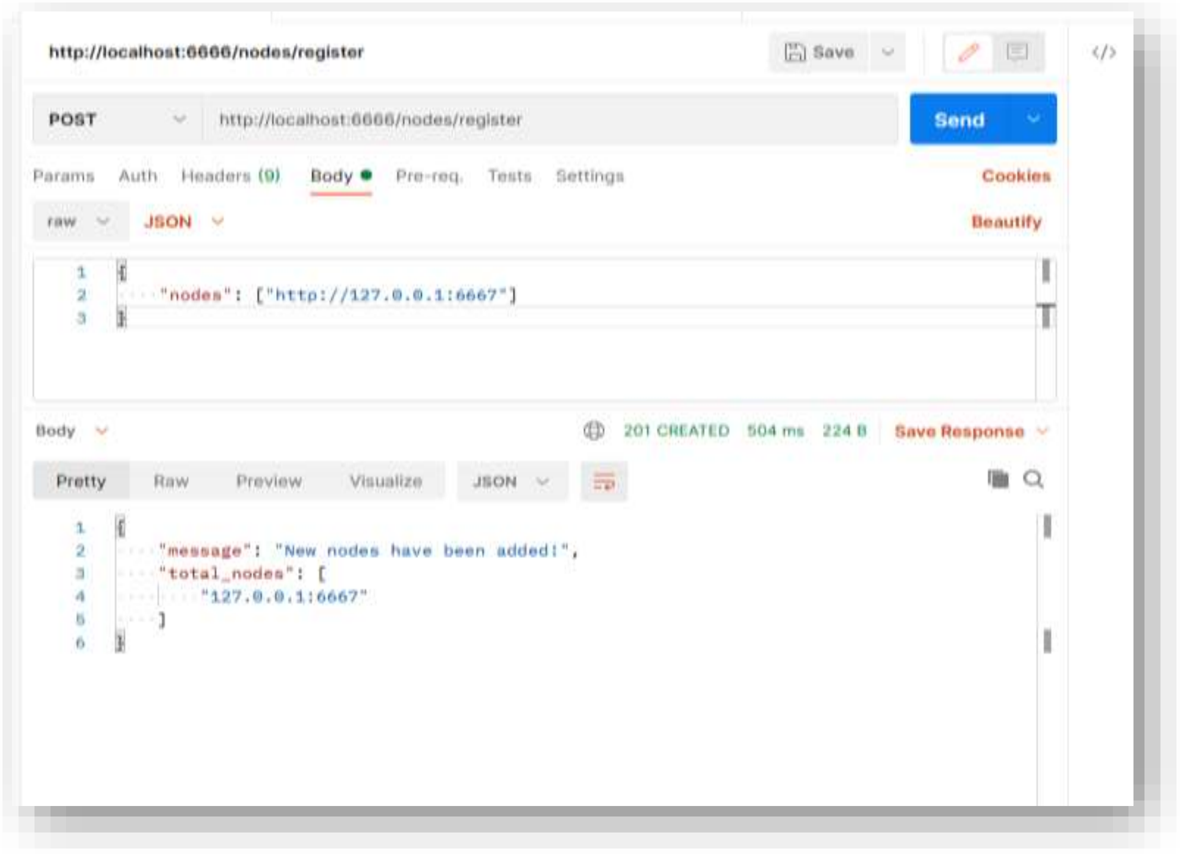
I then added two new blocks to svr 6666:



```
{
    "chain": [
        {
            "index": 1,
            "prev_hash": 1,
            "proof": 100,
            "timestamp": 1611861543.4305956,
            "transactions": []
        },
        {
            "index": 2,
            "prev_hash": "a94b82e2cec4a9bc446c57e44dfcd1c2eafe2c35a7db7078edb2fe507205e881",
            "proof": 35293,
            "timestamp": 1611861751.9611044,
            "transactions": [
                {
                    "amount": 1,
                    "recipient": "8679f6c81ef4495c8a18dd6f6c216350",
                    "sender": "0"
                }
            ]
        },
        {
            "index": 3,
            "prev_hash": "3d7afaa5e1088c4549a270505ef198b5c0c93e8c9d5fc8264d82ce855aeb231b",
            "proof": 35089,
            "timestamp": 1611861773.4835627,
            "transactions": [
                {
                    "amount": 1,
                    "recipient": "8679f6c81ef4495c8a18dd6f6c216350",
                    "sender": "0"
                }
            ]
        }
    ],
    "length": 3
}
```

Figure 72: svr 6666 new blocks

Registered each server's node address to the other:



http://localhost:6666/nodes/register

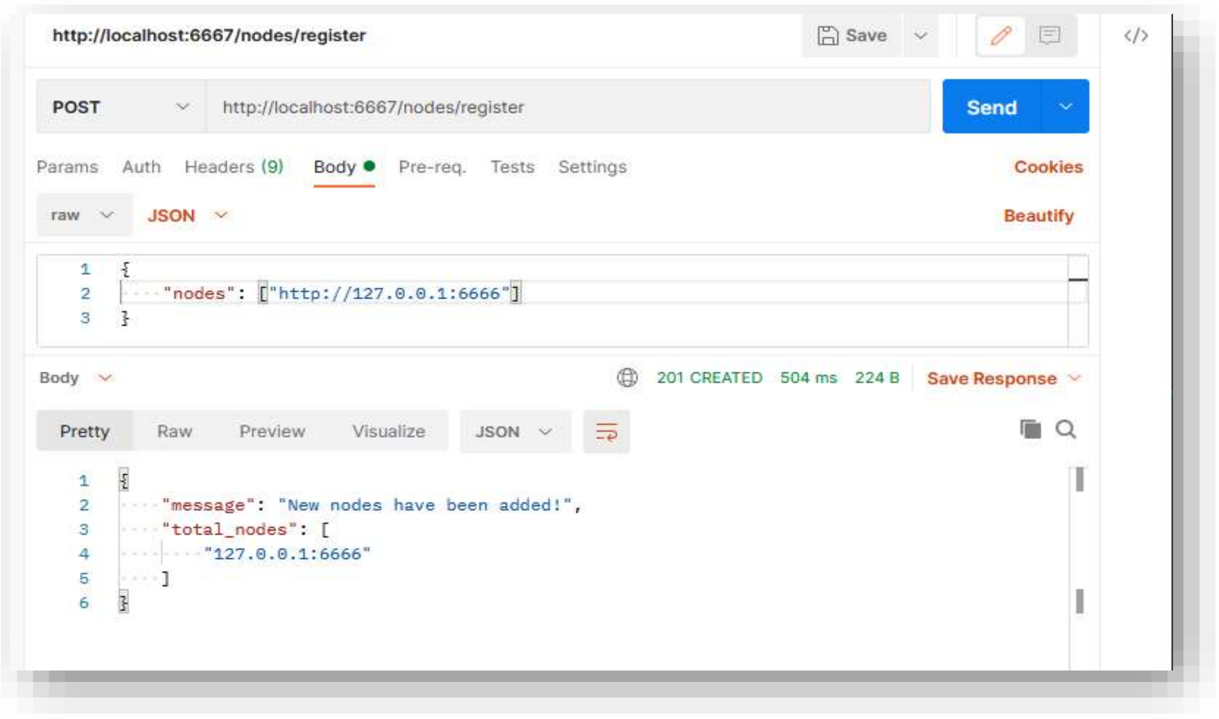
POST http://localhost:6666/nodes/register

```
{
    "nodes": ["http://127.0.0.1:6667"]
}
```

201 CREATED 504 ms 224 B

```
{
    "message": "New nodes have been added!",
    "total_nodes": [
        "127.0.0.1:6667"
    ]
}
```

Figure 73: 6666 reg 6667



http://localhost:6667/nodes/register

POST http://localhost:6667/nodes/register

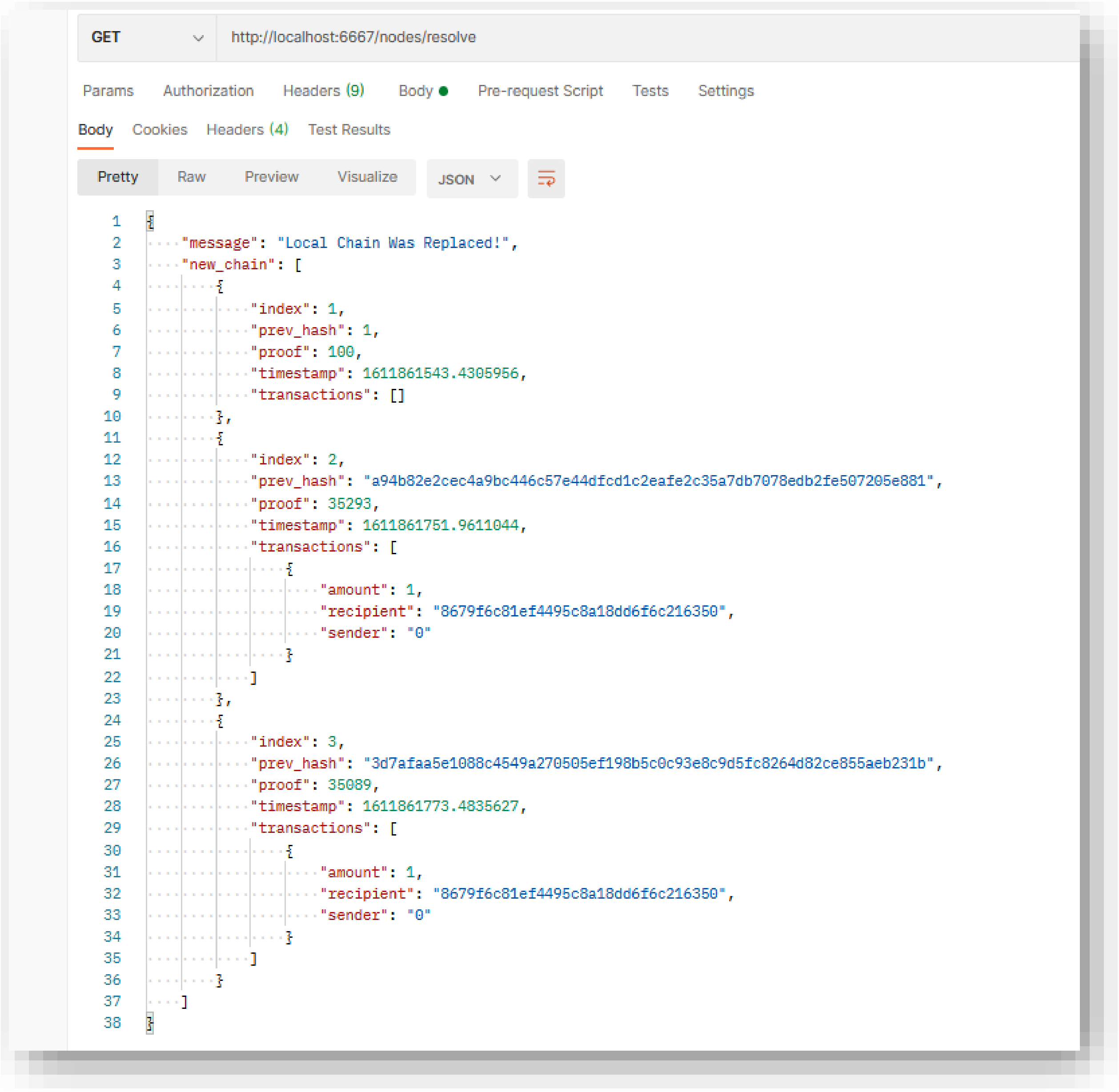
```
{
    "nodes": ["http://127.0.0.1:6666"]
}
```

201 CREATED 504 ms 224 B

```
{
    "message": "New nodes have been added!",
    "total_nodes": [
        "127.0.0.1:6666"
    ]
}
```

Figure 74: 6667 reg 6666

I then ran the resolve on svr 6667 (as its blockchain is shorter) to replace its chain with the longer one present on svr 6666:

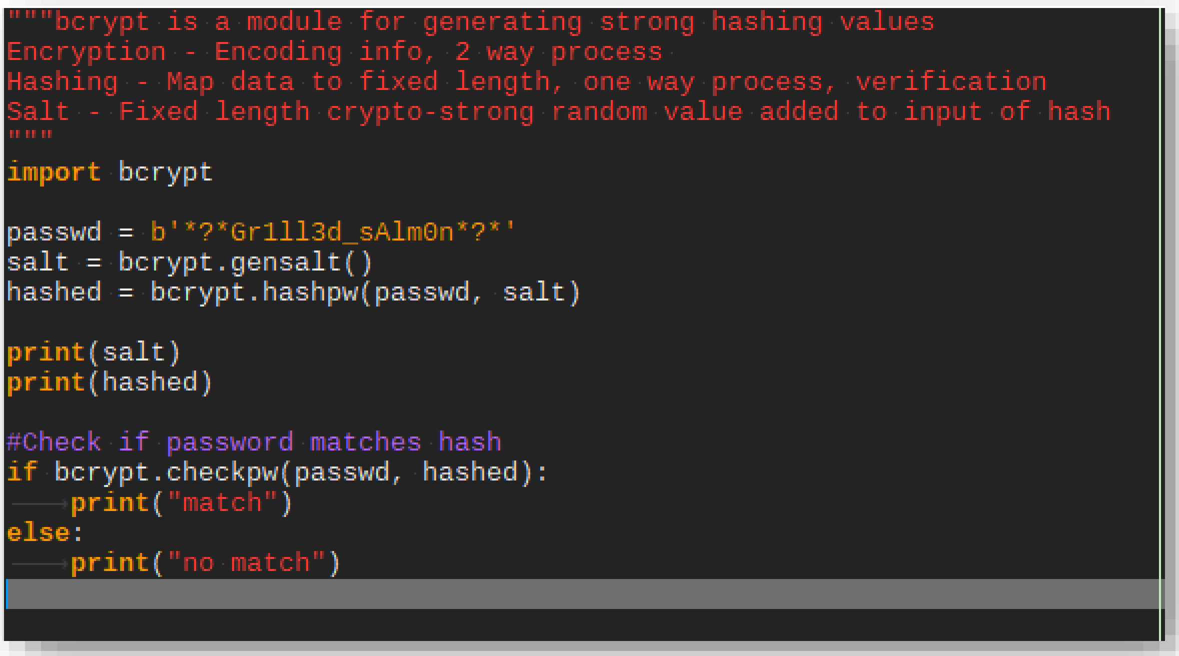


```
http://localhost:6667/nodes/resolve
```

```
{
    "message": "Local Chain Was Replaced!",
    "new_chain": [
        {
            "index": 1,
            "prev_hash": 1,
            "proof": 100,
            "timestamp": 1611861543.4305956,
            "transactions": []
        },
        {
            "index": 2,
            "prev_hash": "a94b82e2cec4a9bc446c57e44dfcd1c2eafe2c35a7db7078edb2fe507205e881",
            "proof": 35293,
            "timestamp": 1611861751.9611044,
            "transactions": [
                {
                    "amount": 1,
                    "recipient": "8679f6c81ef4495c8a18dd6f6c216350",
                    "sender": "0"
                }
            ]
        },
        {
            "index": 3,
            "prev_hash": "3d7afaa5e1088c4549a270505ef198b5c0c93e8c9d5fc8264d82ce855aeb231b",
            "proof": 35089,
            "timestamp": 1611861773.4835627,
            "transactions": [
                {
                    "amount": 1,
                    "recipient": "8679f6c81ef4495c8a18dd6f6c216350",
                    "sender": "0"
                }
            ]
        }
    ]
}
```

Figure 75: svr 6667 after resolve

Hashed Database Passwords: It is very unsecure to store passwords in a database in plaintext format, ideally the passwords would be stored hashed and then when a form is submitted with the login details, the password entered by the user would be hashed and checked against the hashed database password. I created a simple test using a password hashing library called bcrypt:

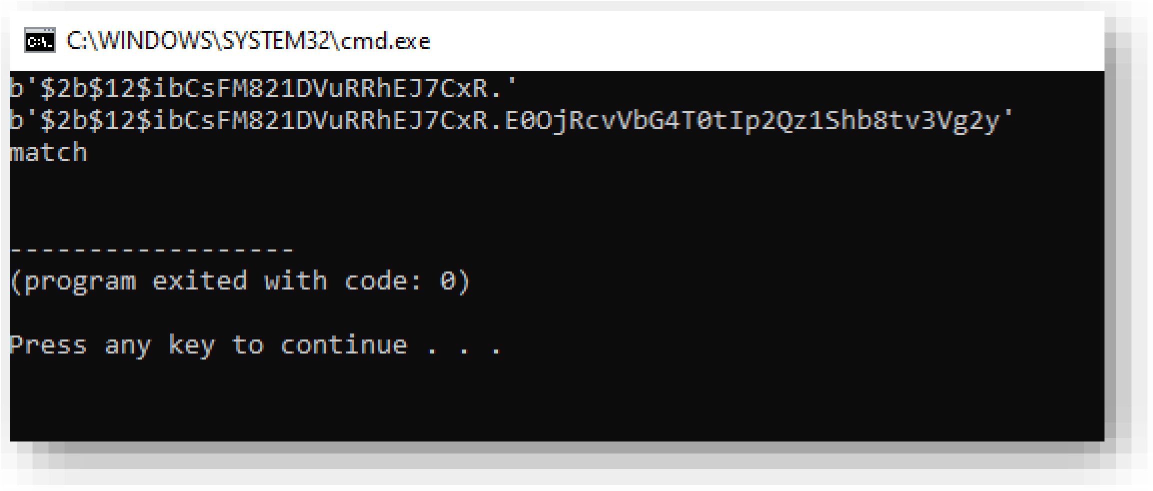


```
"""bcrypt is a module for generating strong hashing values
Encryption - Encoding info, 2 way process
Hashing - Map data to fixed length, one way process, verification
Salt - Fixed length crypto-strong random value added to input of hash
"""
import bcrypt

passwd = b'*?*Gr1ll3d_sAlm0n*?*'
salt = bcrypt.gensalt()
hashed = bcrypt.hashpw(passwd, salt)

print(salt)
print(hashed)

#Check if password matches hash
if bcrypt.checkpw(passwd, hashed):
    print("match")
else:
    print("no match")
```

Figure 76: hash password test

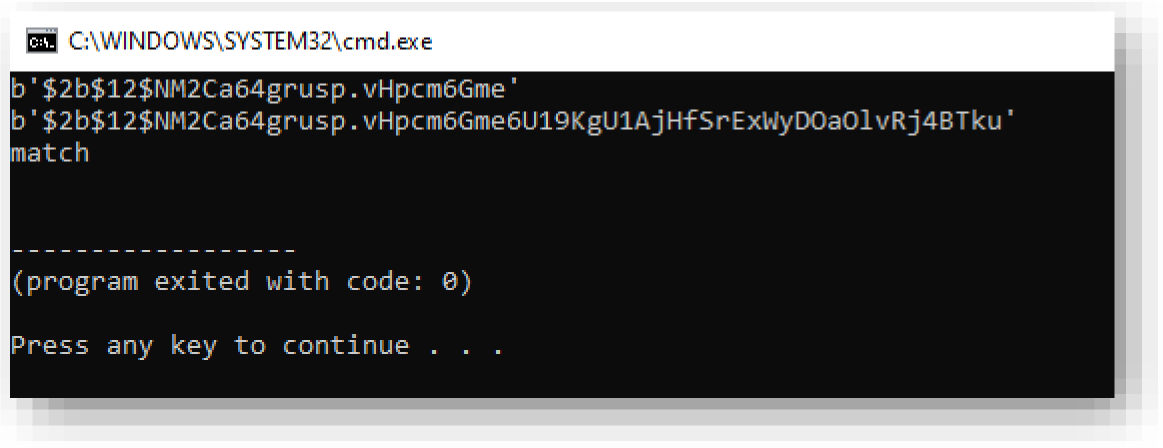
Ideally we would run a user's password text (bytes) through this program and store the result in the database password field, then when we send the password value entered on the login page we would run a similar hash function to match it.

However because a unique salt value is generated each time we get different results:



```
C:\WINDOWS\SYSTEM32\cmd.exe
b'$2b$12$ibCsFM821DVuRRhEJ7CxR.'
b'$2b$12$ibCsFM821DVuRRhEJ7CxR.E0OjRcvVbG4T0tIp2Qz1Shb8tv3Vg2y'
match


------------------
(program exited with code: 0)

Press any key to continue . . .
```

Figure 77: hash password test result 1



```
C:\WINDOWS\SYSTEM32\cmd.exe
b'$2b$12$NM2Ca64grusp.vHpcm6Gme'
b'$2b$12$NM2Ca64grusp.vHpcm6Gme6U19KgU1AjHfSrExWyDOaOlvRj4BTku'
match


------------------
(program exited with code: 0)

Press any key to continue . . .
```

Figure 78: hashed password test result 2

This would make it impossible to get the same hash value for the same password and in turn won't match anything to the database user passwords.

We could overcome this by either using a static salt value or no salt at all (However this decreases password security as two identical passwords would produce identical hashes, meaning if one is compromised, the other is as well). The built-in bcrypt.checkpw() can be used to check if a password matches the hash value, the problem is getting it to work between the database, HTML, and our blockchain program.

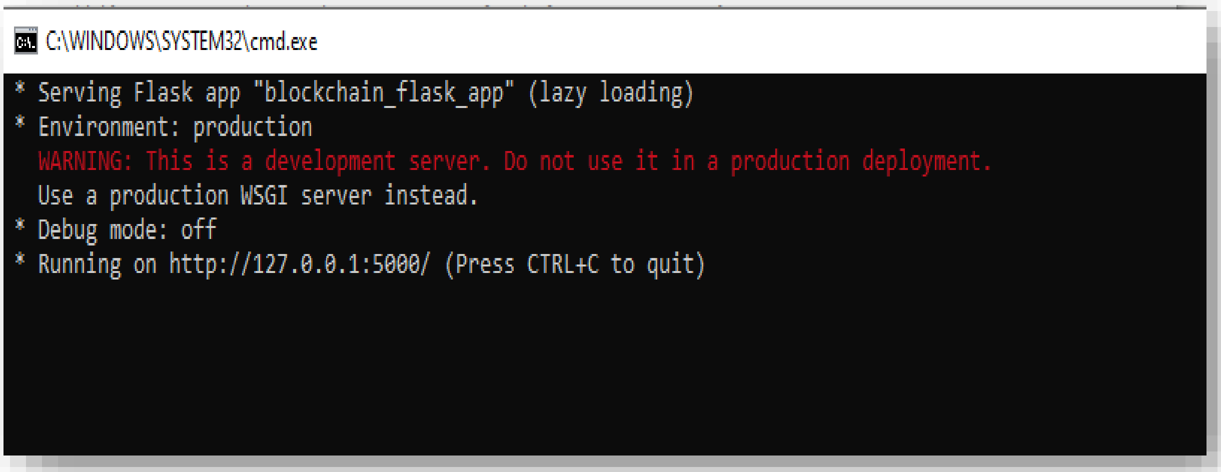
Sessions: As it stands there is no login session establishment on the flask server, this means that anyone can access any resource of the server via it’s URL, if we were going forward with the web interface we would need to setup login sessions on the flask webserver so that only authorized entities could access the add record and add block functionality.

Search Function: I would like to add a sophisticated search engine integrated into the page (instead of ctrl+f) so users would have an easier time navigating through the blockchain. Possibly filter out results in some way based on model number, chassis number etc...

Conclusion

The result of the project is we have a working proof of concept in the form of a solid blockchain app infrastructure that is modular in design. This framework can be easily modified to suit any blockchain-type service; however, additional functionality would be required (outlined in Future of Project section) in order to deploy to a production environment. I will demonstrate the end result of the project as follows:

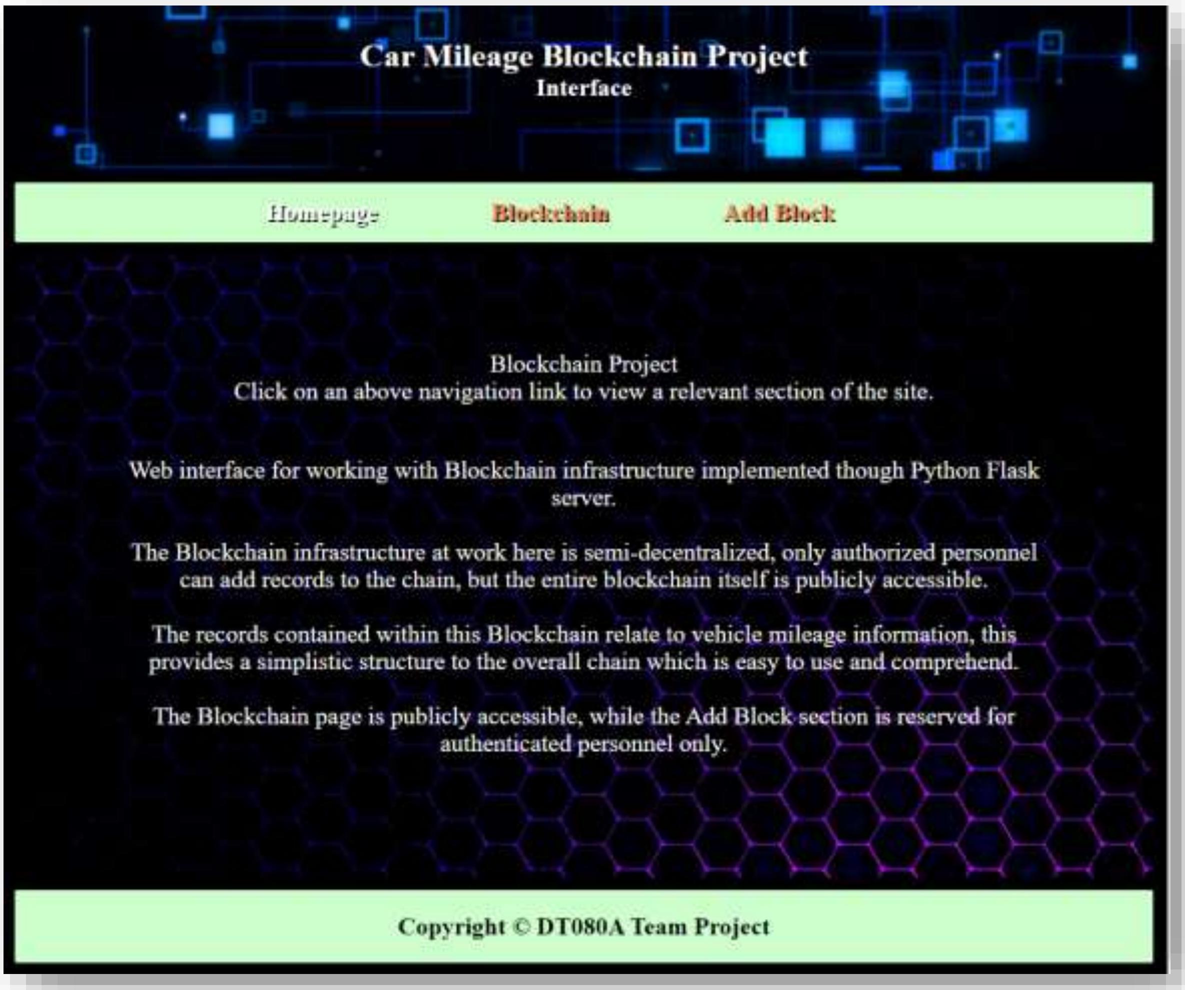
We first start our Blockchain Program (It's assumed we already have the database populated with user entries). If no parameters are added to the flask app.run() method the default launch location is localhost on port 5000:



```
C:\WINDOWS\SYSTEM32\cmd.exe
* Serving Flask app "blockchain_flask_app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figure 79: End server start

If we then access localhost:5000 (or 127.0.0.1:5000) through a web browser we get our homepage:


Car Mileage Blockchain Project
Interface

Blockchain Project
Click on an above navigation link to view a relevant section of the site.

Web interface for working with Blockchain infrastructure implemented though Python Flask server.

The Blockchain infrastructure at work here is semi-decentralized, only authorized personnel can add records to the chain, but the entire blockchain itself is publicly accessible.

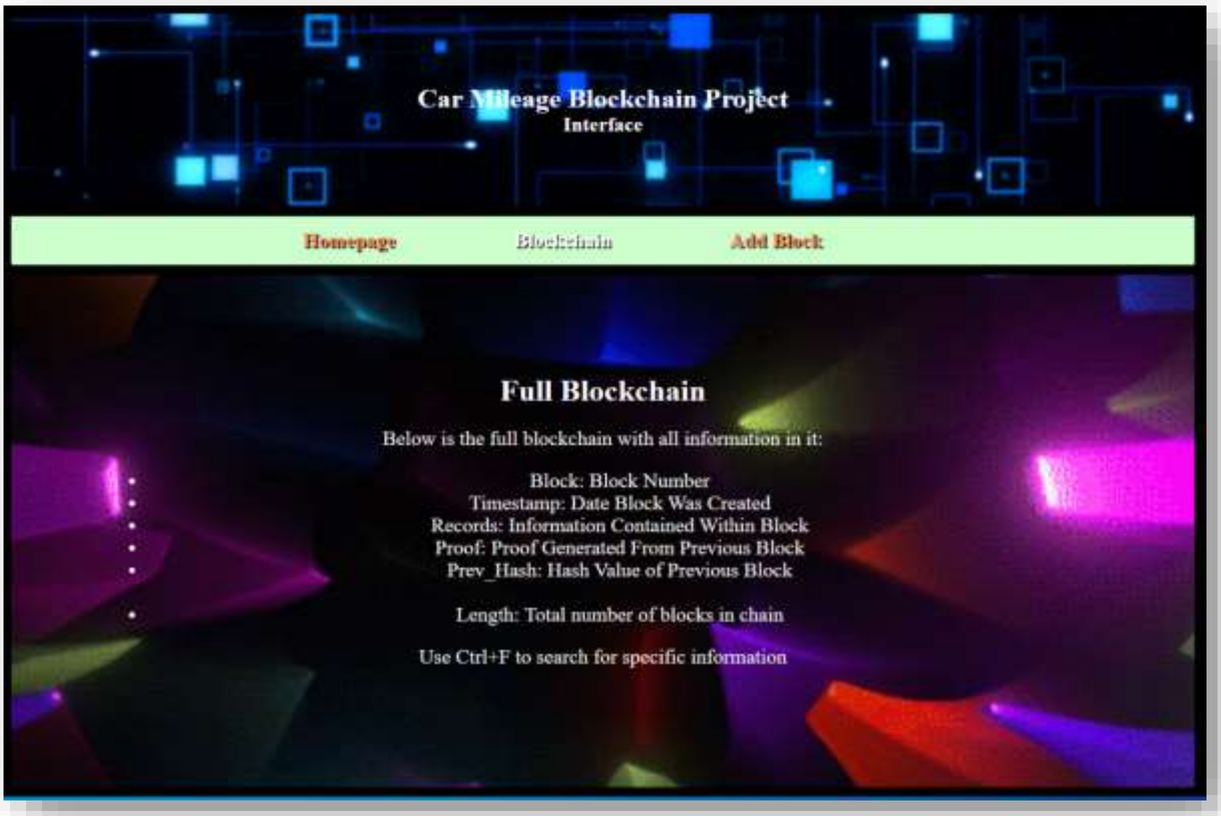
The records contained within this Blockchain relate to vehicle mileage information, this provides a simplistic structure to the overall chain which is easy to use and comprehend.

The Blockchain page is publicly accessible, while the Add Block section is reserved for authenticated personnel only.

Copyright © DT080A Team Project

Figure 80: End Homepage

We can also view the blockchain itself, only the genesis block will be present at Start:



Full Blockchain

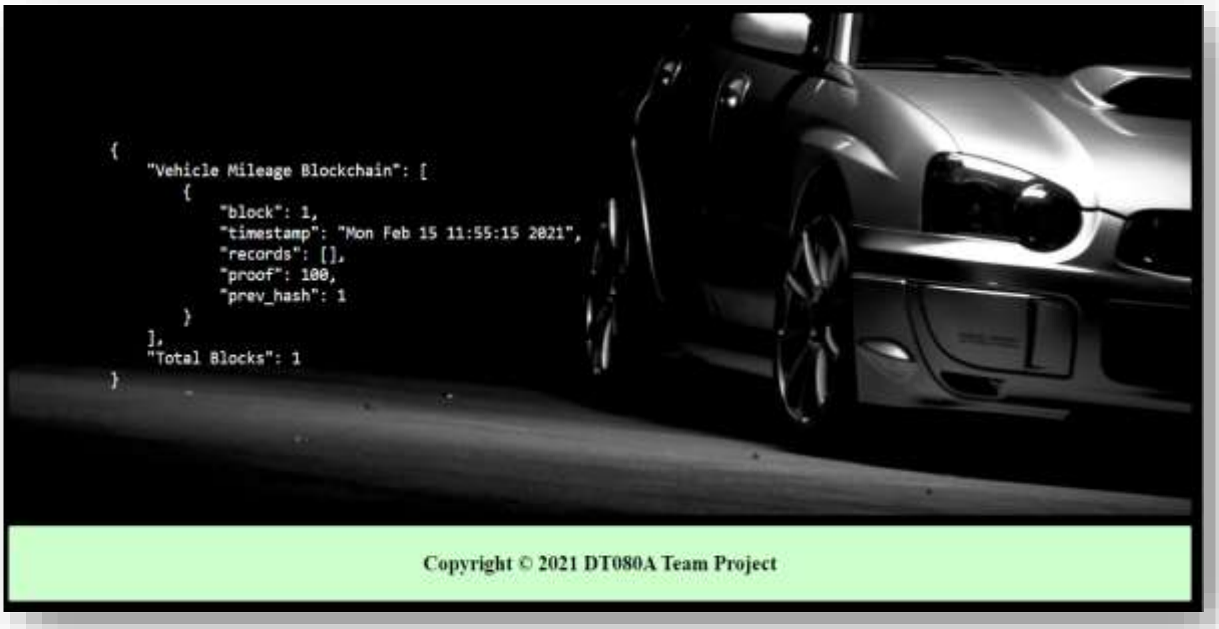
Below is the full blockchain with all information in it:

- Block: Block Number
- Timestamp: Date Block Was Created
- Records: Information Contained Within Block
- Proof: Proof Generated From Previous Block
- Prev_Hash: Hash Value of Previous Block

- Length: Total number of blocks in chain

Use Ctrl+F to search for specific information

Figure 81: End chain 1



```
{
    "Vehicle Mileage Blockchain": [
        {
            "block": 1,
            "timestamp": "Mon Feb 15 11:55:15 2021",
            "records": [],
            "proof": 100,
            "prev_hash": 1
        }
    ],
    "Total Blocks": 1
}
```

Copyright © 2021 DT080A Team Project

Figure 82: End chain 2

We can then login (using admin secret) to add new information to the chain:

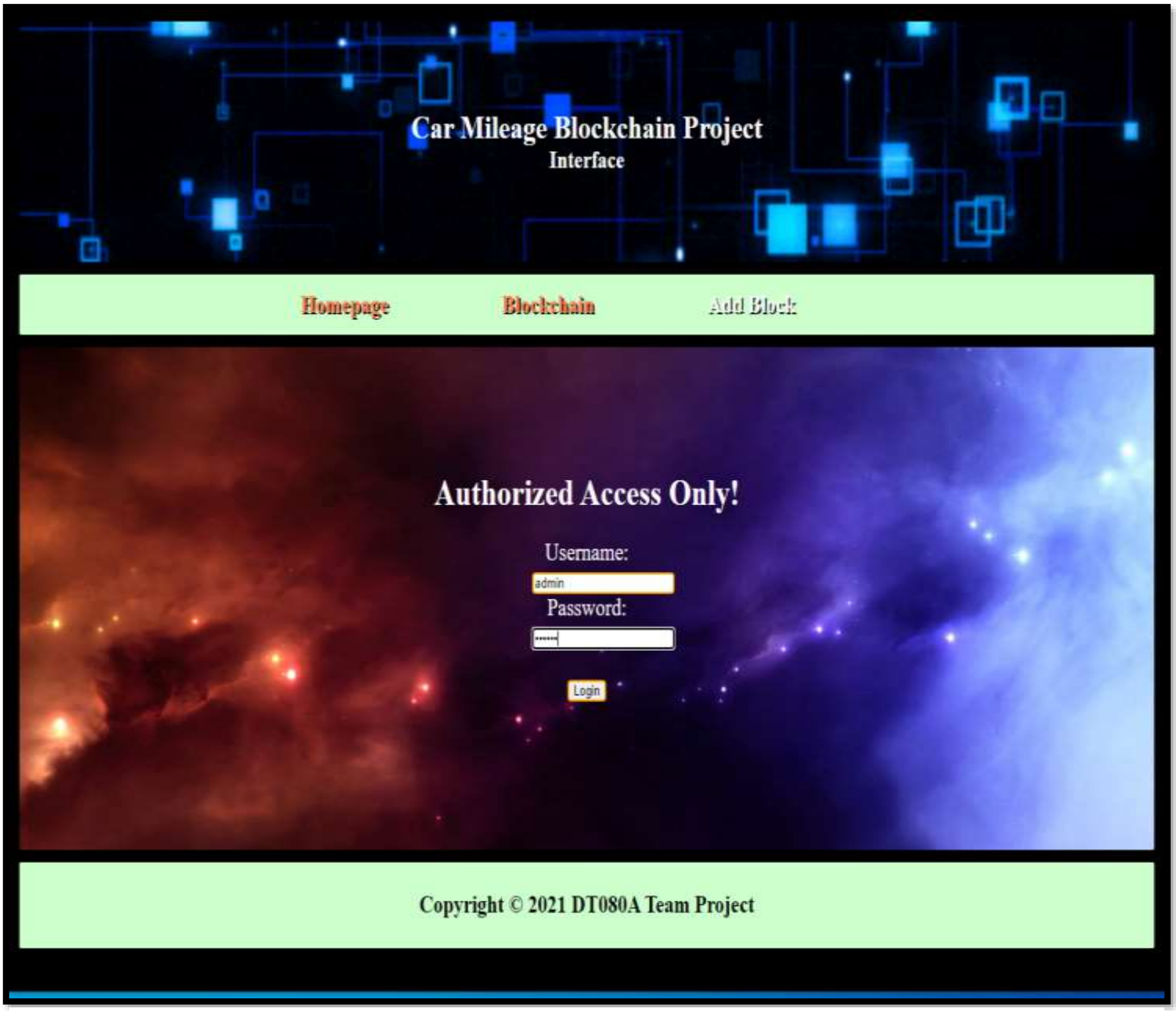
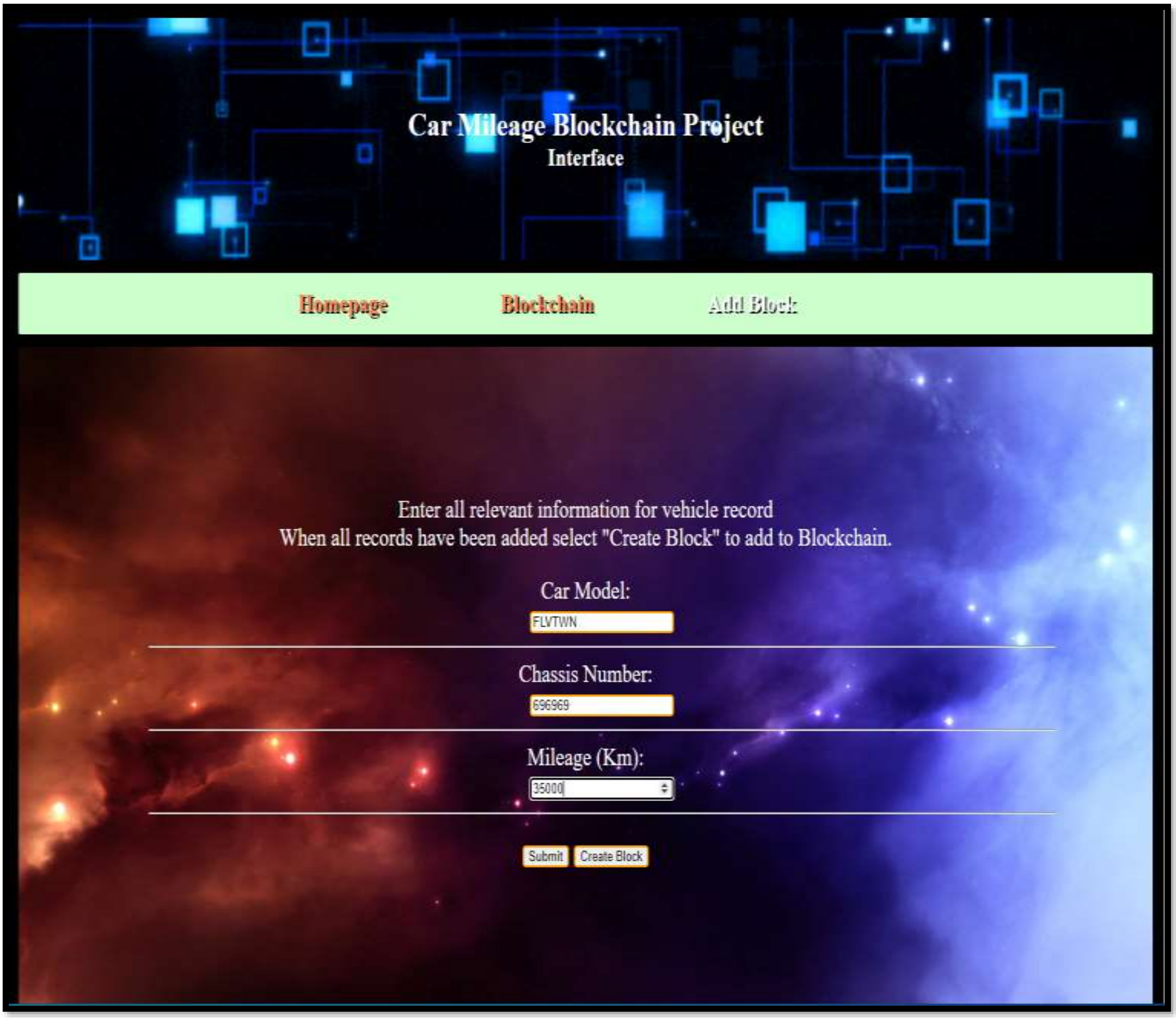


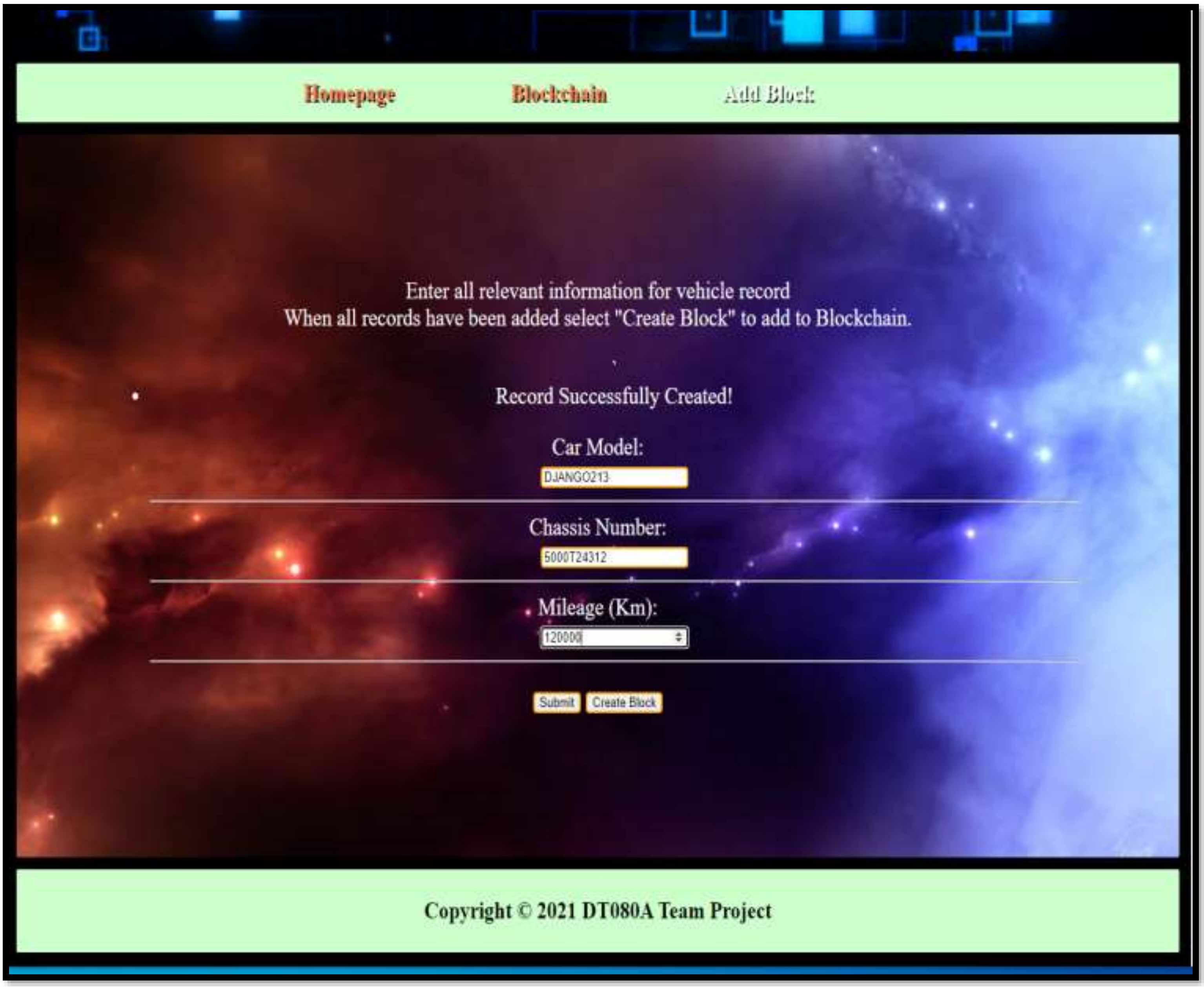
Figure 83: End login

We will then add two new records to the current block:


Car Mileage Blockchain Project
Interface

Enter all relevant information for vehicle record
When all records have been added select "Create Block" to add to Blockchain.
Car Model: FLVTWN
Chassis Number: 696969
Mileage (Km): 35000

Figure 84: End record 1



Enter all relevant information for vehicle record
When all records have been added select "Create Block" to add to Blockchain.

Record Successfully Created!

Car Model:
DJANGO213

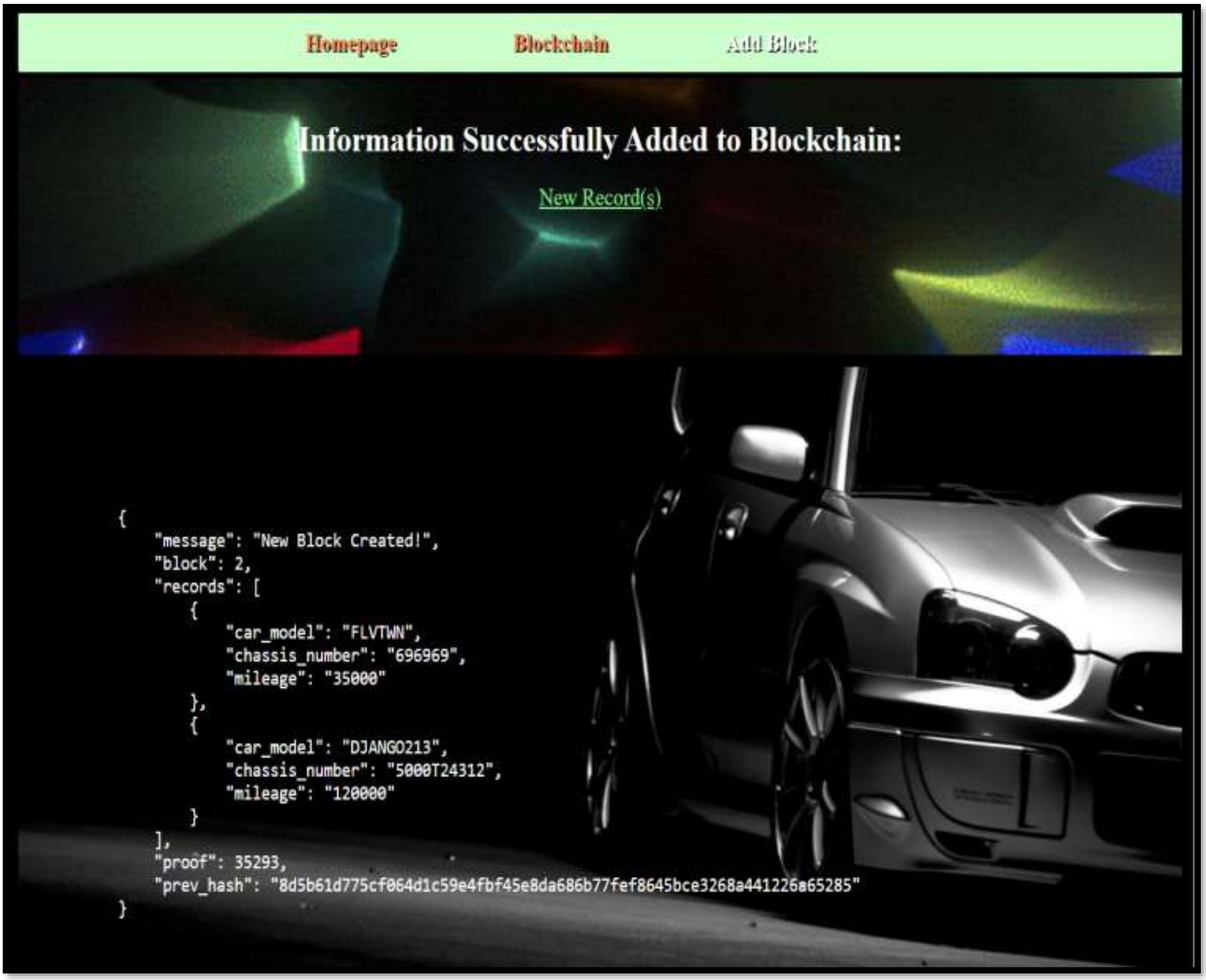
Chassis Number:
5000T24312

Mileage (Km):
120000

Copyright © 2021 DT080A Team Project

Figure 85: End record 2

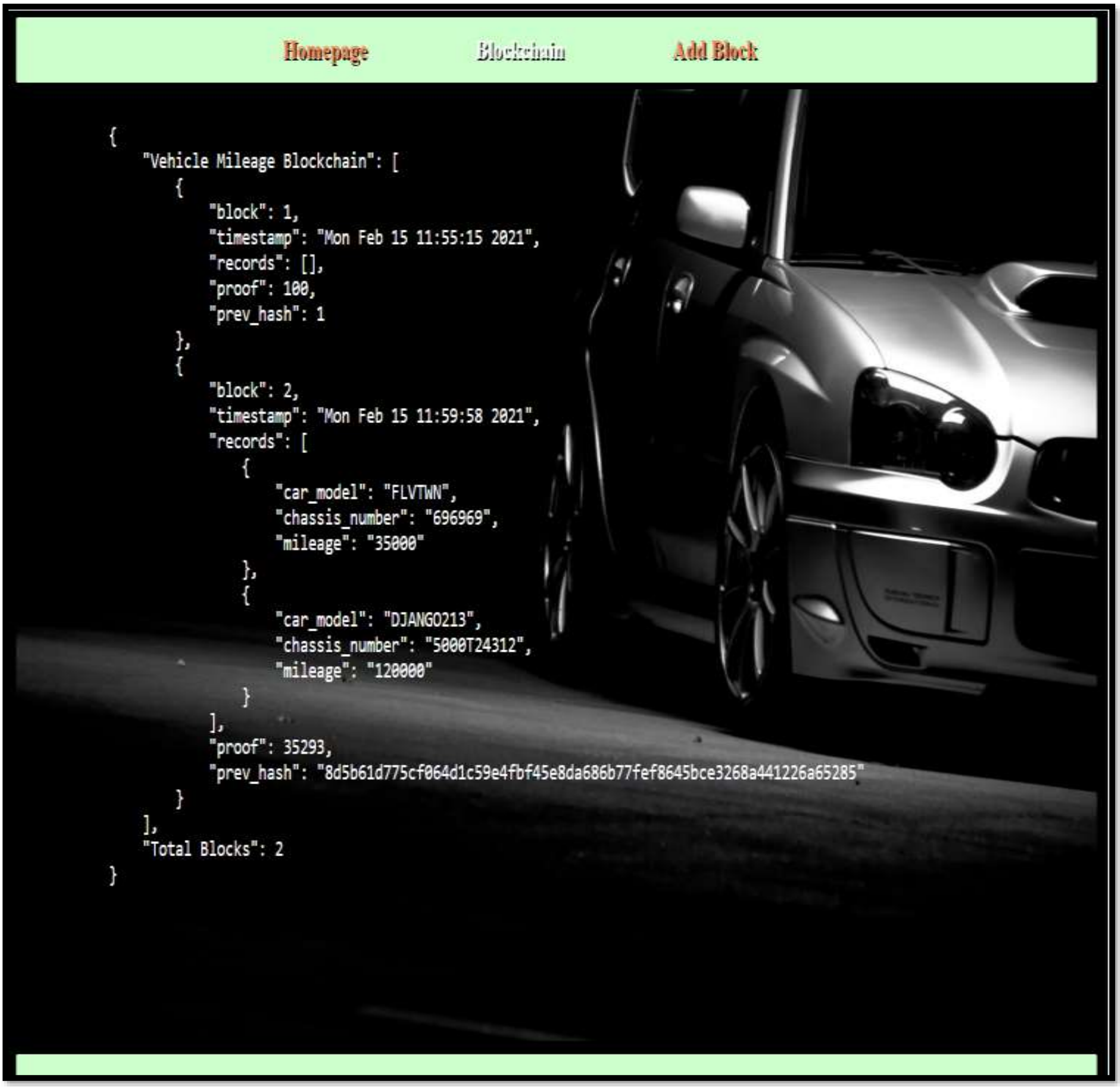
We can now create a new block with these created records by selecting “Create Block”, this will also show the records added to the block:



```
{
    "message": "New Block Created!",
    "block": 2,
    "records": [
        {
            "car_model": "FLVTWN",
            "chassis_number": "696969",
            "mileage": "35000"
        },
        {
            "car_model": "DJANGO213",
            "chassis_number": "5000T24312",
            "mileage": "120000"
        }
    ],
    "proof": 35293,
    "prev_hash": "8d5b61d775cf064d1c59e4fbf45e8da686b77fef8645bce3268a441226a65285"
}
```

Figure 86: End add block

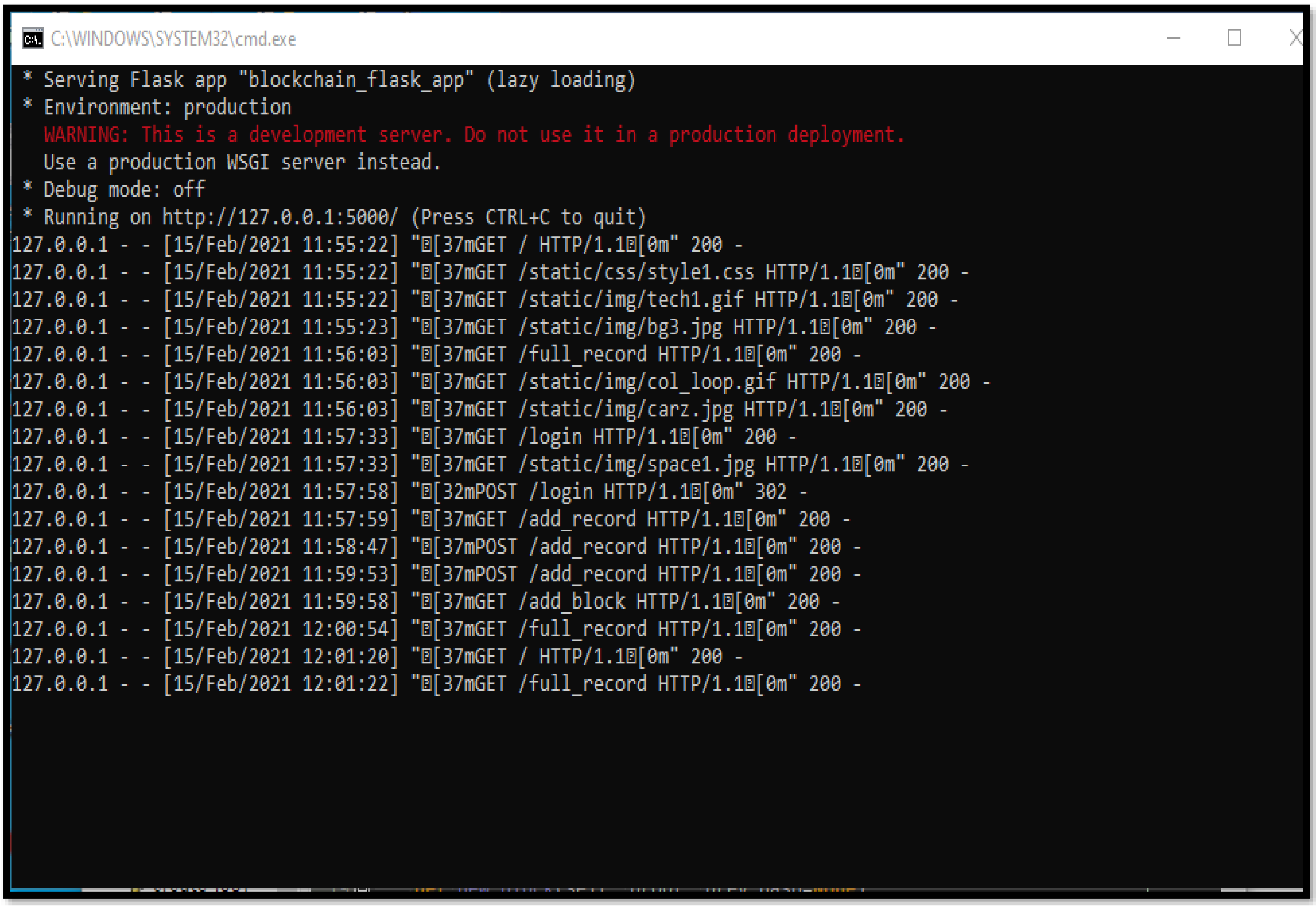
If we go back to the blockchain page we can see this newly added information:



```
{
    "Vehicle Mileage Blockchain": [
        {
            "block": 1,
            "timestamp": "Mon Feb 15 11:55:15 2021",
            "records": [],
            "proof": 100,
            "prev_hash": 1
        },
        {
            "block": 2,
            "timestamp": "Mon Feb 15 11:59:58 2021",
            "records": [
                {
                    "car_model": "FLVTWN",
                    "chassis_number": "696969",
                    "mileage": "35000"
                },
                {
                    "car_model": "DJANGO213",
                    "chassis_number": "5000T24312",
                    "mileage": "120000"
                }
            ],
            "proof": 35293,
            "prev_hash": "8d5b61d775cf064d1c59e4fbf45e8da686b77fef8645bce3268a441226a65285"
        }
    ],
    "Total Blocks": 2
}
```

Figure 87: End new chain

We can also view the end server access log that corresponds to the changes we made:



```
 * Serving Flask app "blockchain_flask_app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [15/Feb/2021 11:55:22] "[37mGET / HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 11:55:22] "[37mGET /static/css/style1.css HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 11:55:22] "[37mGET /static/img/tech1.gif HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 11:55:23] "[37mGET /static/img/bg3.jpg HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 11:56:03] "[37mGET /full_record HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 11:56:03] "[37mGET /static/img/col_loop.gif HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 11:56:03] "[37mGET /static/img/carz.jpg HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 11:57:33] "[37mGET /login HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 11:57:33] "[37mGET /static/img/space1.jpg HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 11:57:58] "[32mPOST /login HTTP/1.1[0m" 302 -
127.0.0.1 - - [15/Feb/2021 11:57:59] "[37mGET /add_record HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 11:58:47] "[37mPOST /add_record HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 11:59:53] "[37mPOST /add_record HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 11:59:58] "[37mGET /add_block HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 12:00:54] "[37mGET /full_record HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 12:01:20] "[37mGET / HTTP/1.1[0m" 200 -
127.0.0.1 - - [15/Feb/2021 12:01:22] "[37mGET /full_record HTTP/1.1[0m" 200 -
```

Figure 88: End server log

References

Link to GitHub Repo:

https://github.com/TU-Dublin-DT080A-3-Team-Project/ominous-penguin

- https://www.freecodecamp.org/news/the-authoritative-guide-to-blockchain-development-855ab65b58bc/ - In-depth intro to blockchain development.
- https://hackernoon.com/learn-blockchains-by-building-one-117428612f46 - Develop blockchain in python tutorial
- https://en.wikipedia.org/wiki/Proof_of_work - Information relating to PoW algorithms.
- https://www.coindesk.com/short-guide-blockchain-consensus-protocols - Blockchain Consensus Protocols
- https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3 - Basic Flask web app tutorial.
- https://www.postman.com/ - API development platform, used in initial GET & POST blockchain testing.
- https://flask.palletsprojects.com/en/1.1.x/ - Python Flask web app framework.
- https://2.python-requests.org/en/master/ - Python requests library.
- https://docs.python.org/3/library/hashlib.html - Python Hashlib docs.
- https://sqlite.org/index.html - SQLite Database Engine.
- https://www.programiz.com/python-programming/methods/built-in/staticmethod - Python @staticmethod decorator
- https://pypi.org/project/bcrypt/ - Python bcrypt.