Taller de Proyecto II

7/12/2020

Informe Final

PS6

INVESTIGACIÓN DE BLOCKCHAIN

Grupo de Desarrollo

- CORSINO ALEXANDRE - 01699/9
- LASCANO HASSEN GAMAL - 01279/2

1. PROYECTO

Introducción

La tecnología blockchain es uno de los más interesantes avances en el campo informático en los últimos tiempos, esta tecnología tiene muchos casos de uso interesantes, como para el uso en ambientes financieros o para el seguimiento de un objeto en el tiempo.

Blockchain, consiste de un registro único, consensuado y distribuido entre varios nodos en una red, los cuales guardan en sí mismos información respecto al bloque mismo, y un vínculo al bloque anterior en la cadena, el cual no es modificable. A medida que se agregan nuevos registros, estos son verificados y validados por la red, y luego añadidos a un nuevo bloque que se enlaza a la cadena. Como base, un bloque tiene un índice, una timestamp que indica el momento en que se crea el bloque, un transacción, la cual contiene información del que envía y recibe el bloque, con la cantidad que se quiera enviar, luego tiene un hash como prueba y un link al hash previo. Cuando se crea una instancia de una blockchain, se la necesita instanciar con un bloque génesis, o sea, el primer bloque, el cual no tiene predecesores. El checkeo utilizado para crear nuevos bloques o minar se llama Proof of work. Proof of work es un algoritmo que intenta descubrir un número que solucione un problema. El número debe ser difícil de encontrar pero fácil de verificar por cualquier nodo en la red. El número a solucionar está atado a condiciones establecidas por el creador de la red, por ejemplo: Que el hash entre dos números termine en 0.

Casos de uso

Control de cadena de suministros, Identidad digital, Mercado inmobiliario, Sector de la salud, Sistemas de votación, Criptomonedas. Control de la cadena de suministros : Todos los activos en la cadena de suministro deberán digitalizarse, pero cada producto, una vez digitalizado, tendrá un número de serie único asociado al mismo. Por lo tanto, cada activo puede ser monitoreado y rastreado a través de la cadena de suministro, en conjunto con otra información asociada al mismo que se quiera conocer. Ejemplo: Food Trust de IBM.

Identidad digital : Forma de tener una fuente confiable para sistemas gubernamentales de verificar la identidad de cierta persona. No es susceptible a robo de identidad de esta manera. Ejemplo: Hyperledger Indy

Sistema de salud: Sirve para el almacenamiento de datos sanitarios de una persona en una red descentralizada, por lo cual se puede tener consistencia entre múltiples hospitales o establecimientos de salud. Ejemplo: SimplyVital healthMercado Inmobiliario : Sirve para el pago de alquileres o compras de propiedades. Ejemplo: propy

Sistema de votación : Usando la identidad única de cada persona, se puede votar de forma segura desde cualquier lugar, llevando a múltiples ventajas: Eliminación del fraude electoral, verificación de votos, almacenamientos de votos seguros en la red como blockchain y aumento de confianza y transparencia. Criptomonedas : El uso tradicional de la tecnología blockchain. Facilita las transacciones entre pares, elimina a posibles intermediarios y sirve como alternativa monetaria a servicios de billetera online. Ejemplos: Bitcoin, Ethereum

Objetivo

El objetivo de este proyecto es realizar una blockchain con el propósito del seguimiento de objetos. Se le asigna un código a un objeto en específico, el cual se podrá escanear para poder recuperar la información actual y pasada del mismo. Asimismo se dispondrá de una página web para poder agregar nuevos objetos a la blockchain. Para enviar los datos de los objetos, contaremos con una placa arduino conectada a una red wifi, la cual manda la información de los objetos a una página web.

Motivacion

La motivación del proyecto es demostrar la función y estructura de una blockchain básica de trazabilidad de objetos, para entender la lógica básica de cada elemento de un sistema con blockchain tal que: nodos (registro y consenso), bloques, transacciones, proof of work, hashing, validación de bloques.

Funciones

Las funciones principales del proyecto son:

- Scan de código rfid asignado a un objeto
- Generar bloques de datos de trazabilidad sobre el objeto
- Generación de transacción entre nodemcu y nodos
- Minar bloques
- Crear consenso, resolver conflictos de blockchain entre nodos
- Agregar mas nodos

2. Materiales y Presupuesto

Mercadolibre

Abajo se encuentra una tabla de los componentes utilizados, fueron comprados por mercadolibre:

COMPONENTE	PRECIO ($ARS)
ESP8266 NODEMCU	569
ARDUINO UNO R3	949
RFID RC522	269
PROTOBOARD	470
CABLES M-M 30cm x10	46.8
CABLE M-H 30cm	7.75
TOTAL	2,311.55

Tab.1

Link facturas:
https://drive.google.com/drive/folders/1FNjfGwtJoEbNF2au5JE7kPxj_9gtItpy?usp=sharing

.

3. Descripción del Proyecto

Esquema

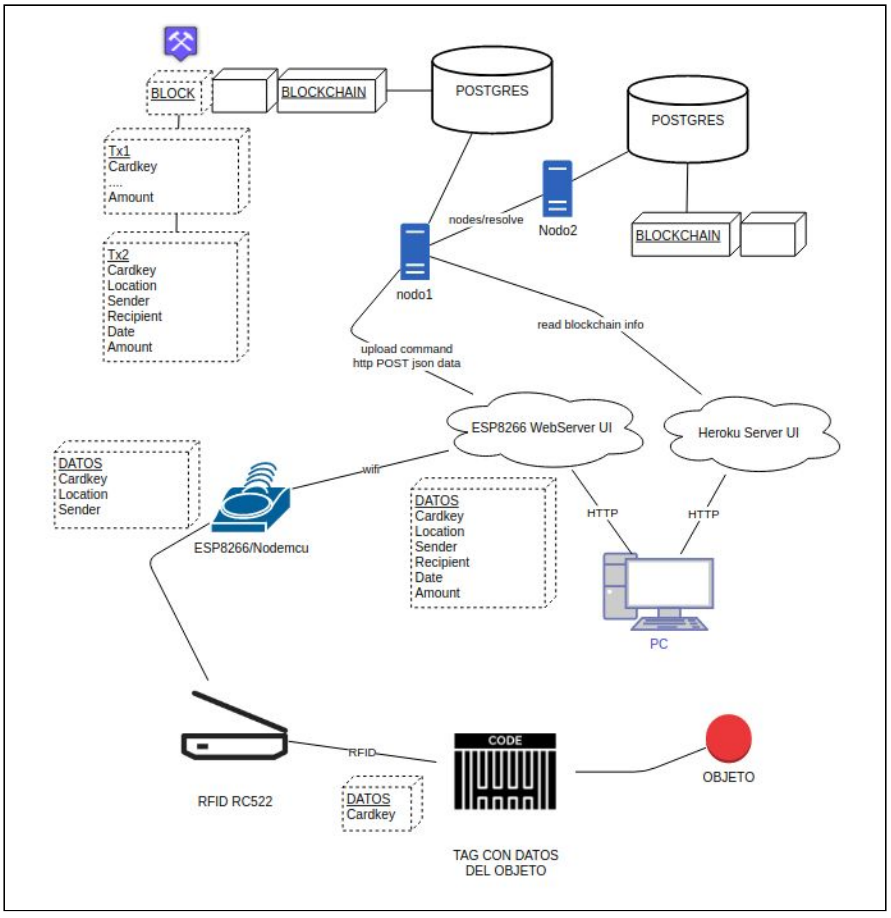


Fig.1: Esquema gráfico del proyecto

Primero asignamos un tag RFID a un objeto y activamos nuestro nodemcu. El nodemcu tiene asignado su punto de acceso (AP) wifi, id, localización(ciudad) y nodo receptor. El AP wifi y nodo receptor pueden ser cambiados por consola. Al iniciar la placa el nodemcu buscará el AP en un loop hasta poder conectarse a este. Luego el nodemcu webserver se iniciará y el usuario podrá entrar en este para ejecutar comandos.

Ejecutando el comando para escanear el objeto en el sensor RFID, la placa nodemcu registra el objeto. Luego podemos ejecutar un comando para generar una transacción, los datos se empaquetan en formato JSON y contienen: id origen, id destinatario (puede ser cualquier nodo), localización, fecha, código rfid y monto de movimiento. Se envía la transacción al nodo receptor mediante un HTTP Post y este lo guardará en su pool de transacciones y lo enviará a los otros nodos.

Al recibir el objeto en la página cliente, se podrá confirmar la transacción minando un nuevo bloque. Hecho esto podemos resolver conflictos entre nodos mandando el comando resolve a otros nodos para que la blockchain más larga y válida sea establecida en este.

Hardware

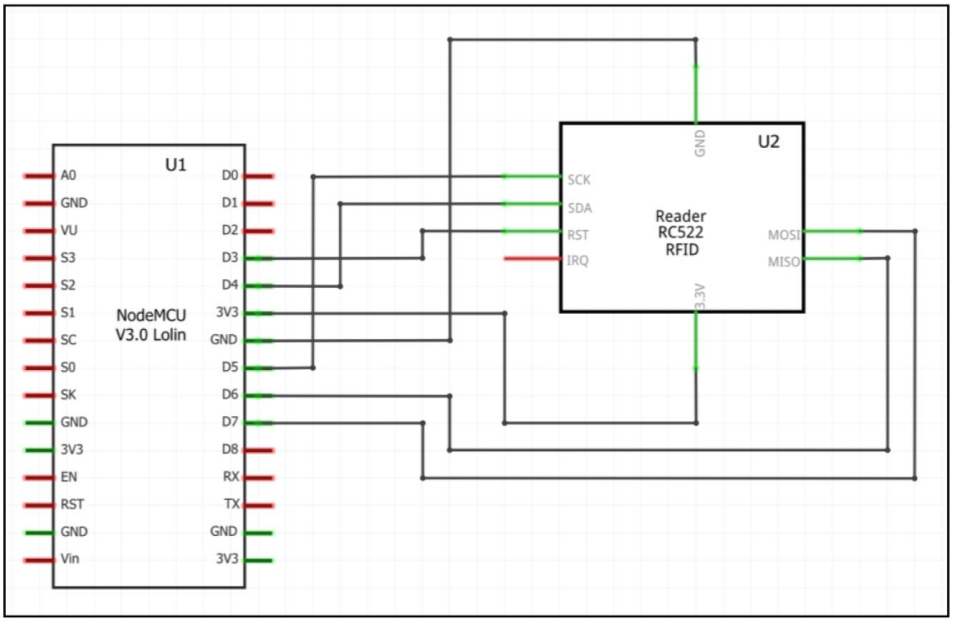


Fig.4: diagrama nodemcu y scanner

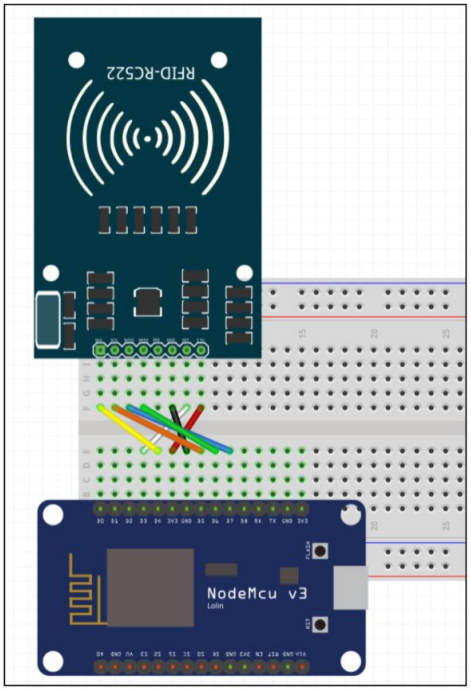


Fig.5: esquema nodemcu y scanner

Alimentacion

Alimentación del ESP8266/Nodemcu (WiFi)

El módulo **ESP8266** es alimentado usando un cable y su puerto mini usb **V5**.

Alimentación del Rfid Rc522

El módulo **Rfid Rc522** es alimentado a través del NodeMcu por el pin **V3.3**

Comunicaciones

a) E/S del controlador/placa de desarrollo con el exterior, excepto PC

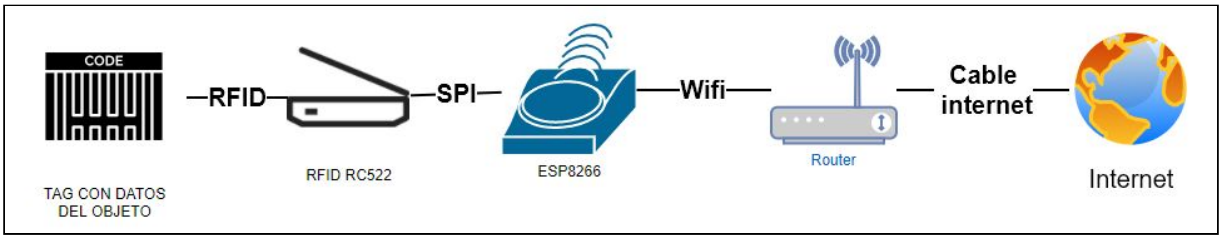


Fig.2: comunicación nodemcu a internet

SPI

El scanner rc522 se comunica con el nodemcu usando el protocolo SPI, permite comunicación sincrónica en un bus entre un maestro y múltiples esclavos.

SCLK es el pin reloj para mandar solo cuando hay un flanco alto de reloj, MOSI y MISO son pines master output slave input y master input slave output para determinar quien envia y quien recibe, SS es slave select para elegir el dispositivo a communicar.

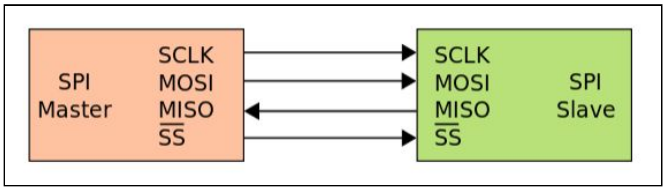


Fig.2b: protocolo SPI

b) Comunicaciones con la PC

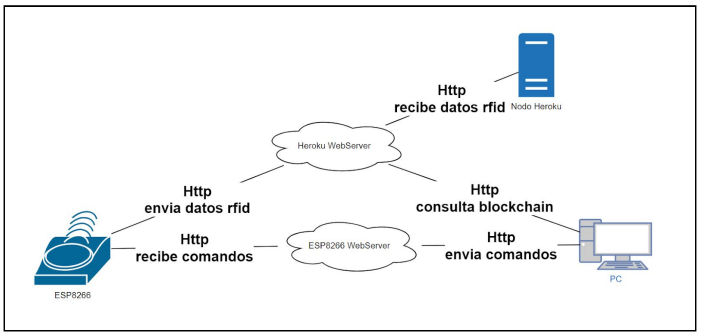


Fig.3: comunicación entre dispositivos

RFID RC522

El scanner rc522 sirve para escanear códigos RFID.

ESP8266

Esta placa nos permite instalar módulos y conectarse a una red wifi y luego internet para poder comunicarse con la pc y el sitio heroku.

HTTP POST y GET

El nodemcu utilizara el protocolo HTTP comando POST para mandar los datos rfid al sitio web heroku y para recibir comandos de la PC lo hará con HTTP GET.

SERVIDOR ESP8266

El sitio web del nodemcu nos permite controlar la placa mandando comandos. Los comandos esenciales son scan para escanear una tarjeta y upload para mandar los datos al sitio heroku.

SERVIDOR NODOS HEROKU

Los sitios web estaran hosteado en Heroku y dispondrán de una base de datos postgres donde se almacenará la blockchain en cada nodo.

c) Sistema web

NODO HEROKU: Sitio web utilizado para el hosting y el mantenimiento del sitio web

NODEMCU SERVER: Sitio web utilizado para permitir la comunicación entre PC y Nodemcu

POSTGRES: Sistema de base de datos utilizado en conjunto con heroku para mantener la base de datos del sitio

PYTHON: Lenguaje de programación utilizado para programar el sitio web y la blockchain que mantiene el sistema

FLASK: Framework utilizado para implementar el sitio web.

Codigo

Nodemcu

Modulo Scanner RFID

La librería MFRC552 nos permite escanear la tarjeta. Cuando se ejecuta la función el scanner espera una tarjeta, el usuario puede mandar el comando exit para anular el scan. Cuando aparece una tarjeta el scanner recupera los datos de esta en su buffer, verifica el tipo del tag y luego empieza a sacar los datos del buffer para guardarlos en la variable cardkey.

```
#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN 2
#define RST_PIN 0

MFRC522 rfid(SS_PIN, RST_PIN); //RFID set pins
MFRC522::MIFARE_Key key; //key
String cardKey = "PLEASE SCAN A CARD"; //cardkey stored here

/**
 * scan the rfid cardkey in cardkey
 */
void scanRfidCard()
{
      Serial.println("\nPlace card in front of the RFID SCANNER");
      Serial.println("type exit to go back to menu");
      menuCmd="";
      //wait for card scan or select
      while ( (!rfid.PICC_IsNewCardPresent() ||
!rfid.PICC_ReadCardSerial() ) && menuCmd != "exit") {
            readUserMenuCmd(); //scan a card or exit
      }
      if(menuCmd != "exit")
      {
```

```
            //CHECK TYPE MIFARE type
            // Serial.print(F("PICC type: "));
            MFRC522::PICC_Type piccType =
rfid.PICC_GetType(rfid.uid.sak);
            // Serial.println(rfid.PICC_GetTypeName(piccType));
            if (piccType != MFRC522::PICC_TYPE_MIFARE_MINI &&
                        piccType != MFRC522::PICC_TYPE_MIFARE_1K &&
                        piccType != MFRC522::PICC_TYPE_MIFARE_4K) {
                  Serial.println(F("Your tag is not of type MIFARE
Classic."));
                  return;
            }

            //SCAN THE CARD
            Serial.print("\nUID tag :");
            String cardKeyScanning= "";
            byte letter;
            for (byte i = 0; i < rfid.uid.size; i++)
            {
                  Serial.print(rfid.uid.uidByte[i] < 0x10 ? " 0" : "-");
                  Serial.print(rfid.uid.uidByte[i], HEX);
                  cardKeyScanning.concat(String(rfid.uid.uidByte[i] <
0x10 ? " 0" : "-"));
                  cardKeyScanning.concat(String(rfid.uid.uidByte[i],
HEX));
            }
            cardKeyScanning.toUpperCase();
            Serial.println();
            rfid.PICC_HaltA();
            rfid.PCD_StopCrypto1();
            cardKey = cardKeyScanning;
      }
}
```

Modulo Wifi

Se utiliza la librería ESP8266Wifi para conectarse a un access point wifi. Tenemos que configurar el ssid y contraseña en la sección wifi settings para conectarse. Guarda el ip en la variable chipid, esta ip nos servirá para entrar en el webserver de la placa.

```
#include <ESP8266WiFi.h>
```

```
//WIFI SETTINGS
String ssid     = "WIFINAME"; //wifi 2.4ghz config   .SETTINGS.
String password = "MYPASS"; //default wifi AP

String chipip; //use wifiinfo to find your webserver ip

/**
 * connect to default wifi - returns true if successful or false if not
 */
boolean connectWifiSetup()
{
      boolean state = true;
      int i = 0;

      WiFi.mode(WIFI_STA); //AP mode
      WiFi.begin(ssid, password); //connecting to defautl wifi
      Serial.println("");
      Serial.println("Connecting to WiFi");

      // Wait for connection
      Serial.print("Connecting ...");
      while (WiFi.status() != WL_CONNECTED) { //retry to connect
multiple times
            delay(500);
            Serial.print(".");
            if (i > 10){
                  state = false;
                  break;
            }
            i++;
      }

      if (state){
            chipip = WiFi.localIP().toString();
            wifiInfo();
      }
      else {
            Serial.println("");
            Serial.println("Connection failed.");
      }
      return state;
}

/**
 * shows the wifi ssid and ip address of the nodemcu
 */
```

```
String wifiInfo()
{
      String wifiinfostr;
      wifiinfostr="";
      wifiinfostr+="\n<br>Connected to ";
      wifiinfostr+=ssid;
      wifiinfostr+="\n<br>IP address: ";
      wifiinfostr+=chipip; //muestra ip asignada del nodemcu
      Serial.println(wifiinfostr);
      return wifiinfostr;
}
```

Webserver Nodemcu

El nodemcu inicia el servidor con *startServer* y se comunica con su webserver con la función *nodemcuWebserver* donde la placa espera la llegada de un usuario al sitio. Luego recupera el comando http del usuario y lo ejecuta. Esta función se encarga de mandar el código html directamente al usuario después de un request http.

```
#include <ESP8266WebServer.h>  //nodemcu webserver
#include <ESP8266HTTPClient.h> //http comm

//NODE SETTINGS
String nodeurl = "http://MYAPP.herokuapp.com/"; //http only, should be
best node for less lag
String recipientAddress = "MYNODEUUID"; //default recipient uuid, can be
any node
int serverport = 80; //WEB SERVER PORT

//WEBSERVER
WiFiServer server(serverport); //nodemcu webserver
String clientHeader;
String clientCharTemp;

String jsonArgs = ""; //json data to send
int serverConnected = false; //nodemcu server on?

/**
 * start the nodemcu webserver.
 */
String startServer() //IN SETUP
{
      String startserverstr;
```

```
      startserverstr="";
      if(wifiConnected)
      {
            startserverstr+="\n<br>Starting server...";
            server.begin();
            serverConnected = 1;
            startserverstr+= "\n<br>Server address: ";
            startserverstr+= chipip;
            startserverstr+= ":";
            startserverstr+= serverport;
      } else {
            startserverstr+="\n<br>Wifi not connected!";
      }
      Serial.println(startserverstr);
      return startserverstr;
}

/**
 * starts the nodemcu webserver. use the nodemcu ip address + port to
access the site in any browser
 */
void nodemcuWebserver()  //IN LOOP
{
      if(serverConnected) {
            //server.handleClient();
            WiFiClient client = server.available();   // Listen for
incoming clients

            if (client) {                             // If a new client
connects,
                  Serial.println("New Client.");          // print a
message out in the serial port
                  String currentLine = "";                // make a
String to hold incoming data from the client
      String currentform = "console";
                  currentTime = millis();
                  previousTime = currentTime;
                  while (client.connected() && currentTime -
previousTime <= timeoutTime) { // loop while the client's connected
                        currentTime = millis();
                        if (client.available()) {             // if
there's bytes to read from the client,
                              char c = client.read();             // read
a byte, then
                              Serial.write(c);                    //
print it out the serial monitor
```

```
                                clientHeader += c;
                                if (c == '\n') {                    // if
the byte is a newline character
                                    // if the current line is blank, you
got two newline characters in a row.
                                    // that's the end of the client HTTP
request, so send a response:
                                    if (currentLine.length() == 0) {
                                          // HTTP clientHeaders always
start with a response code (e.g. HTTP/1.1 200 OK)
                                          // and a content-type so the
client knows what's coming, then a blank line:
                                          client.println("HTTP/1.1 200
OK");

client.println("Content-type:text/html");
                                          client.println("Connection:
close");
                                          client.println();

              //clean our client console if too full
              if(strlen(clientCharTemp.c_str()) > 512)
              {
                clientCharTemp="";
              }


                                          //print data in our homemade
console
                                          if (clientHeader.indexOf("GET
/get?console=menu") >= 0) {
                                                clientCharTemp+=
showMenu();
                                          } else if
(clientHeader.indexOf("GET /get?console=wifiinfo") >= 0) {
                                                clientCharTemp+=
wifiInfo();
                                          } else if
(clientHeader.indexOf("GET /get?console=server") >= 0) {
                                                clientCharTemp+=
startServer();
                                          } else if
(clientHeader.indexOf("GET /get?console=scan") >= 0) {
                                                scanRfidCard();
                                                clientCharTemp+= "<br>
Scan command completed";
                                          } else if
```

```
(clientHeader.indexOf("GET /get?sendto=") >= 0) {

                                                //parse http arguments
                const char* clientHeaderstr = clientHeader.c_str();
                                                char *b =
strstr(clientHeaderstr, "sendto="); //find pos of args, sendto= is 7
char
                                                int pos = b -
clientHeaderstr + 7;

                                                String args =
clientHeaderstr + pos;
                char* argsstr;
                strcpy(argsstr,args.c_str());
                                                char** tokens;
                                                tokens =
str_split(argsstr, '_');


recipientAddress=*(tokens);
                                                nodeurl=*(tokens + 1);
                                                free(*(tokens));
                                                free(*(tokens+1));
                                                free(*(tokens+2));
                                                free(tokens);

                                                clientCharTemp+=
"<br>CHANGING WEBSITE TO: ";
                                                clientCharTemp+= nodeurl;
                                                clientCharTemp+=
"<br>CHANGING RECIPIENT TO: ";
                                                clientCharTemp+=
recipientAddress;
                                        } else if
(clientHeader.indexOf("GET /get?console=upload") >= 0) {
                                                writeTx();
                                                sendTx();
                                                clientCharTemp+= "<br>
Transaction command completed";

                                                clientCharTemp+= "<br>
Data sent to " + nodeurl;
                                                clientCharTemp+= "<br>";
                                                clientCharTemp+=
jsonArgs;
                                        }
```

```
                                        // Display the HTML web page
                                        // this part can be cleaned up
with using SPIFFS and html files, check out:

//https://tttapa.github.io/ESP8266/Chap11%20-%20SPIFFS.html
                                        client.println("<!DOCTYPE
html><html>");
                                        client.println("<head><meta
name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
                                        client.println("<link
rel=\"icon\" href=\"data:,\"></head>");

                                        // Web Page Heading

client.println("<body><h1>ESP8266 Web Server</h1>");

                                        // content
                                        client.println("<p>Last scan:
"+cardKey+"</p>");
              client.println("<p>command: "+currentform+"</p>");
                                        client.println("<form
action=\"/get\"><input type=\"text\"
name=\""+currentform+"\"><br><br>");
                                        client.println("<input
type=\"submit\" value=\"Submit\"></form>");

client.println("<br>_____________________________________________________
_____________________________________<br>");

client.println("<p>"+clientCharTemp+"</p>");

client.println("<br>_____________________________________________________
_____________________________________<br></body></html>");

                                        // The HTTP response ends with
another blank line
                                        client.println();
                                        // Break out of the while loop
                                        break;
                                    } else { // if you got a newline,
then clear currentLine
                                        currentLine = "";
                                    }
                                } else if (c != '\r') {  // if you got
anything else but a carriage return character,
```

```
                                    currentLine += c;      // add it to
the end of the currentLine
                                }
                            }
                        }
                        // Clear the clientHeader variable
                        clientHeader = "";
                        // Close the connection
                        client.stop();
                        Serial.println("Client disconnected.");
                        Serial.println("");
                    }
                }
}
```

Transaccion

El nodemcu y su webserver pueden hacer transacciones y mandarlas a nodos. Para esto se formatean los datos de transacción en json y se mandan al sitio nodo con un http post.

```
#include <ESP8266WebServer.h>  //nodemcu webserver
#include <ESP8266HTTPClient.h> //http comm
#include <ArduinoJson.h> //json

/**
 * prepare the transaction(tx) data in json format. parameters:
 * sender, recipient, amount, cardkey, location, date
 */
void writeTx()
{
  jsonArgs="";
      Serial.println("\nWriting JSON...");
      StaticJsonDocument<512> doc; //json lib v6
      doc["sender"] = chipid; //chip address
      doc["recipient"] = recipientAddress; //send to node address
      doc["amount"] = 1; //all amounts can be added to show how many
times card was scanned
      doc["cardkey"] = cardKey;
      doc["location"] = location;
      doc["date"] = datetimenow(); //unix time
      serializeJson(doc,jsonArgs);
      Serial.println(jsonArgs);
```

```
      Serial.println("JSON DONE");
      //jsonCardKey = "{\"Cardkey\":\"" + cardKey + "\"}";
}

/**
 * send tx to website
 */
void sendTx()
{
      HTTPClient http;
      String heroku_thumbprint;
      String httpTempData;
  String fullurl;
  fullurl = nodeurl;
      httpTempData="transactions/new"; //add card url
      //httpTempData+= httpArgs; //add card variable
      fullurl+= httpTempData;  //full url

      Serial.print("connecting to ");
      Serial.println(fullurl.c_str());
      http.begin(fullurl.c_str()); //connect to website
      http.addHeader("content-type","application/json"); //add http
header, we set content to send in json
      int httpCode = http.POST(jsonArgs); //send the data to website

      //we get the http response code
      if(httpCode == HTTP_CODE_OK) {
            Serial.printf("httpcode: %d\n",httpCode);
            httpTempData = http.getString(); //receive data
            Serial.print("httpTempData");
            Serial.println(fullurl);
      }
      else {
            Serial.printf("error code: %d\n",httpCode);
      }
      http.end();
}
```

Nodo Heroku

Blockchain

Variables de la blockchain

```
class Blockchain(object):
```

```
def __init__(self):
self.chain = []
self.current_transactions = []
self.new_block(proof=100, previous_hash=1)
self.nodes = set()
```

Nuestra blockchain tiene como variables la lista de bloques, las transacciones no confirmadas (o mempool del nodo), el primer bloque (genesis) con proof y previous hash asignados y un set de nodos que usaran la blockchain.

Cada nuevo bloque contiene el hash del bloque anterior. Es importante porque es esto que nos da la inmutabilidad de la blockchain. Si un atacador corrompe un bloque anterior entonces todos los bloques que siguen tendrán hashes incorrectos. Explicaremos en detalle esto en la sección proof of work.

Transacciones

Simplemente una función que nos sirve para guardar la información que queremos en la blockchain, en este caso, datos de trazabilidad de un objeto. Recupera los parámetros de entrada, los estructura en formato diccionario de python (similar al formato json) y agrega esta estructura en la lista de transacciones no confirmadas.

Código para crear transacciones

```
    @property
    def last_block(self):
        # Returns the last Block in the chain
        return self.chain[-1]

    def new_transaction(self, sender, recipient, amount, cardkey,
location, date):
        # Adds a new transaction to the list of transactions
        """
        Creates a new transaction to go into the next mined Block
        :param sender: <str> Address of the Sender
        :param recipient: <str> Address of the Recipient
        :param amount: <int> Amount
        :param cardkey: <str> Cardkey
        :param location: <str> Location
        :param date: <str> Date
        :return: <int> The index of the Block that will hold this
transaction
        """
```

```
        self.current_transactions.append({
            'sender': sender,
            'recipient': recipient,
            'amount': amount,
            'cardkey': cardkey,
            'location': location,
            'date': date,
        })

        return self.last_block['index'] + 1
```

Nodos

Esta función nos permite agregar nodos que hostean una copia de la blockchain. Registra los urls de los nodos en un set. Esto nos servirá después para resolver conflictos entre nodos si la copia de la blockchain de un nodo es distinta a otro.

Le pasamos la url de un nodo a la función y esta cambiará el formato de la url para agregarla al set. Primero con *urlparse* para sacar las componentes de la url en 6 tuplas de acorde al estándar RFC 1808: *scheme://netloc/path;parameters?query#fragment*
ejemplo:

```
urlparse('http://www.cwi.nl:80/%7Eguido/Python.html')
ParseResult(scheme='http', netloc='www.cwi.nl:80',
path='/%7Eguido/Python.html',
            params='', query='', fragment='')
```

Luego agrega la parte netloc (network locality) al set.

```
def register_node(self, address):
"""
Add a new node to the list of nodes
:param address: <str> Address of node. Eg. 'http://192.168.0.5:5000'
:return: None
"""
parsed_url = urlparse(address)
self.nodes.add(parsed_url.netloc)
```

Proof of Work

Un algoritmo proof of work sirve para crear nuevos bloques en la cadena. El propósito es encontrar un numero solución que resuelve un problema. El número debe ser

difícil de encontrar pero muy fácil de verificar (analogía. Cubo rubik). Así los nodos podrán verificar la blockchain rápidamente.

Ejemplo básico:

```
Let's decide that the hash of some integer x multiplied by another y
must end in 0. So,
    hash(x * y) = ac23dc...0
And for this simplified example, let's fix
    x = 5
Implementing this in Python:
from hashlib import sha256
x = 5
y = 0  # We don't know what y should be yet...
while sha256(f'{x*y}'.encode()).hexdigest()[-1] != "0":
    y += 1
print(f'The solution is y = {y}')

The solution here is
    y = 21
Since, the produced hash ends in 0:
    hash(5 * 21) = 1253e9373e...5e3600155e860
Source:
https://hackernoon.com/learn-blockchains-by-building-one-117428612f46
```

Para nuestro algoritmo proof of work, tenemos que encontrar un número p para que cuando se hashea con la solución del bloque anterior, un hash de cuatros 0s iniciales se produzca.

Para ajustar la dificultad se pueden agregar la cantidad de 0s. Basta con agregar un 0 para aumentar significativamente la complejidad del problema.

Para implementar nuestro algoritmo proof of work usamos la librería hashlib.

HASH

Un algoritmo hash utiliza datos y utiliza estos datos para crear un hash o firma. Un ejemplo de un algoritmo hash (malo) es tener un archivo como datos (con números enteros y texto por ejemplo) y un algoritmo hash que recupera todos los dígitos dentro y los suma para generar el hash. Una buena función hash tiene que ser libre de colisiones, difícil de recuperar el dato a partir del hash (hiding) y puzzle friendly. Antes se usaba MD5 pero tiene muchas colisiones entonces usamos SHA256.

SHA: Secure Hashing Algorithm

La función sha256(datosIn) nos permite crear un hash SHA-256 a partir de datos de entrada que llamaremos datosIn.

Tenemos un input (datosIn) que apunta a bits que parecen ser aleatorios de tamaño fijo (hash). Estos bits pueden ser caracteres hexadecimal o binario por ejemplo y pueden servir como firmas.

'Input' -> 01101100

El proceso SHA empieza con un estado interno, traemos los bits de nuestros datosIn 1 a la vez en una función de compresión, luego sumamos el resultado de esto con el estado interno y la salida de esta suma vuelve a ser el nuevo estado interno, cuando los datos terminan el estado interno es el hash.

Nuestro estado interno (H) tiene tamaño 256bits (son 8 Hs. H0 a H7). SHA funciona con bloques de 512bits, hay que hacer padding sino el proceso nos dará overflow, el padding agrega 0s y utiliza los últimos bits del bloque para guardar la longitud del bloque sin los 0s.

Luego tenemos una función de compresión, la seguridad del hash depende de esta. El trabajo de esta función es mezclar una palabra de datosIn N rondas.

El resultado de la función de compresión nos da datos comprimidos, ahora sumamos los Hs con estos datos comprimidos. Esta suma nos da un resultado y será nuestro nuevo H, repetimos este proceso hasta terminar todas las palabras de datosIn. El estado interno H final es el hash de tamaño fijo que necesitamos, el resultado de una función hash se puede llamar digest.

Nuestra validación de prueba toma el proof del bloque anterior y el proof actual y los concatena en un string utf-8 con encode, luego empieza el hashing dando un hash hexadecimal. Si ese hash empieza con cuatros 0s es válido.

F-strings es similar a str.format() nos permite poner las variables proof como strings.

```
  def proof_of_work(self, last_proof):
        """
        Simple Proof of Work Algorithm:
         - Find a number p' such that hash(pp') contains leading 4
zeroes, where p is the previous p'
```

```
         - p is the previous proof, and p' is the new proof
        :param last_proof: <int>
        :return: <int>
        """

        proof = 0
        while self.valid_proof(last_proof, proof) is False:
            proof += 1

        return proof

    @staticmethod
    def valid_proof(last_proof, proof):
        """
        Validates the Proof: Does hash(last_proof, proof) contain 4
leading zeroes?
        :param last_proof: <int> Previous Proof
        :param proof: <int> Current Proof
        :return: <bool> True if correct, False if not.
        """

        guess = f'{last_proof}{proof}'.encode()
        guess_hash = hashlib.sha256(guess).hexdigest()
        return guess_hash[:4] == "0000"
```

Creación de bloque

Luego de terminar la ejecución de nuestra función proof of work se crea un nuevo bloque para confirmar todas las transacciones no confirmadas en este bloque.

El bloque guarda sus variables de índice, proof, prevhash, tiempo de creacion y puntero a su lista de transacciones.

```
    @staticmethod
    def hash(block):
        # Hashes a Block
        """
        Creates a SHA-256 hash of a Block
        :param block: <dict> Block
        :return: <str>
        """
        # We must make sure that the Dictionary is Ordered, or we'll
have inconsistent hashes
        block_string = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(block_string).hexdigest()
```

```
    def new_block(self, proof, previous_hash=None):
        # Creates a new Block and adds it to the chain
        """
         Create a new Block in the Blockchain
         :param proof: <int> The proof given by the Proof of Work
algorithm
         :param previous_hash: (Optional) <str> Hash of previous Block
         :return: <dict> New Block
         """

        block = {
            'index': len(self.chain) + 1,
            'timestamp': time(),
            'transactions': self.current_transactions,
            'proof': proof,
            'previous_hash': previous_hash or self.hash(self.chain[-1]),
        }

        # Reset the current list of transactions
        self.current_transactions = []

        self.chain.append(block)
        return block
```

Funciones de nodos

Un nodo puede verificar si su blockchain es válida verificando que el hash del bloque previo guardado en un bloque siguiente corresponde al recálculo del hash del bloque previo. Iterando así hasta terminar la cadena.

Nuestro algoritmo de consenso puede resolver conflictos en nuestra blockchain comparándola con la de otros nodos y guardando la blockchain más larga en nuestro nodo. El url elNodo/chain de un nodo nos devuelve la longitud y blockchain de elNodo así que utilizamos http get para iterar la blockchain de cada nodo.

```
   def valid_chain(self, chain):
        """
        Determine if a given blockchain is valid loop and verify hash
and proof
        :param chain: <list> A blockchain
        :return: <bool> True if valid, False if not
        """
```

```
        last_block = chain[0]
        current_index = 1

        while current_index < len(chain):
            block = chain[current_index]
            print(f'{last_block}')
            print(f'{block}')
            print("\n-----------\n")
            # Check that the hash of the block is correct
            if block['previous_hash'] != self.hash(last_block):
                return False

            # Check that the Proof of Work is correct
            if not self.valid_proof(last_block['proof'],
block['proof']):
                return False

            last_block = block
            current_index += 1

        return True

    def resolve_conflicts(self):
        """
        This is our Consensus Algorithm, it resolves conflicts
        by replacing our chain with the longest one in the network.
        loops nodes. if valid and longer its choosen
        :return: <bool> True if our chain was replaced, False if not
        """

        neighbours = self.nodes
        new_chain = None

        # We're only looking for chains longer than ours
        max_length = len(self.chain)

        # Grab and verify the chains from all the nodes in our 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 if the length is longer and the chain is valid
```

```
                if length > max_length and self.valid_chain(chain):
                    max_length = length
                    new_chain = chain

        # Replace our chain if we discovered a new, valid chain longer
than ours
        if new_chain:
            self.chain = new_chain
            return True

        return False
```

Sitio

Aca esta el codigo del archivo flask que contiene las rutas del sitio. Al tope del archivo tenemos librerías hashlib para gestionar los hashes, json para formatear datos para la cadena y comunicaciones, datetime y time para recuperar el tiempo y fecha, uuid para que nuestro nodo tenga una dirección de envío (node_identifier), flask para nuestra aplicación, request para comunicarnos con http, jsonify para formatear datos json, render_template para renderizar nuestros archivos html.

```
import hashlib
import json
import time
import datetime
from textwrap import dedent
from uuid import uuid4
import sys #args
from models.Blockchain import *
from models.Cardkey import *
from flask import Flask, jsonify, request, render_template

#args port
#if (len(sys.argv) == 3):
#    myport = int(sys.argv[1])
#else:
#    myport = 80
myport = 80
myhost = "ps6taller.herokuapp.com"

# Instantiate our Node
app = Flask(__name__)
```

```
# Generate a globally unique address for this node
node_identifier = str(uuid4()).replace('-', '')

# Instantiate the Blockchain
blockchain = Blockchain()

# context for all routes
@app.context_processor
def inject_uuid():
    context = {'nodeuuid': node_identifier}
    return context

###
###ROUTES ACA
###

if __name__ == '__main__':
    app.run(host=myhost, port=myport)
```

Rutas

Estas rutas permiten al nodemcu y otros nodos interactuar entre ellos.
/chain nos devuelve la blockchain y su longitud, se llama cuando un nodo quiere resolver conflictos. Un usuario puede ir a esta dirección para visualizar la blockchain entera de este nodo.

/transaction/new permite agregar una transacción no confirmada. El nodemcu llama este routing para mandar la información de trazabilidad del objeto escaneado. El nodo también lo usa para hacer un broadcast de la transacción a los otros nodos.

/mine es la routa que permite minar un bloque para confirmar las transacciones. Crea un bloque con una transacción inicial. También agrega el bloque en nuestra base de datos.

/node/register permite agregar un nodo en el set de nodos de nuestro sitio.

/nodes/resolve empieza el algoritmo de resolución de conflicto

```
@app.route('/chain', methods=['GET'])
def full_chain():
    response = {
```

```
        'chain': blockchain.chain,
        'length': len(blockchain.chain),
    }
    return jsonify(response), 200

# add tx enpoinsd
@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    values = request.get_json()

    # Check that the required fields are in the POST'ed data
    required = ['sender', 'recipient', 'amount', 'cardkey', 'location',
'date']
    if not all(k in values for k in required):
        return 'Missing values', 400

    # Create a new Transaction
    index = blockchain.new_transaction(
        values['sender'], values['recipient'], values['amount'],
values['cardkey'], values['location'], values['date'])

    response = {'message': f'Transaction will be added to Block {index},
broadcasting to other nodes'}

   # find a way to broadcast tx to nodes
   # mutliple response, use socket? or multiple route
   # headers = {'Content-type': 'application/json; charset=UTF-8'}
   # for anode in blockchain.nodes
   #     nodeurl = anode
   #     response = requests.post(nodeurl, data=values, headers=headers)

    return jsonify(response), 201

# mining endpoint
# Calculate the Proof of Work
# Reward the miner (us) by adding a transaction granting us 1 coin
# Forge the new Block by adding it to the chain


@app.route('/mine', methods=['GET'])
def mine():
    # We run the proof of work algorithm to get the next proof...
    last_block = blockchain.last_block
    last_proof = last_block['proof']
    proof = blockchain.proof_of_work(last_proof)
```

```
    # We must receive a reward for finding the proof.
    # The sender is "0" to signify that this node has mined a new coin.
    blockchain.new_transaction(
        sender="0",
        recipient=node_identifier,
        amount=0,  #no reward
        cardkey=0,
        location=0,
        date=str(datetime.datetime.now()).split('.')[0], #unix time
    )

    # Forge the new Block by adding it to the chain
    previous_hash = blockchain.hash(last_block)
    block = blockchain.new_block(proof, previous_hash)

    response = {
        'message': "New Block Forged",
        'index': block['index'],
        'transactions': block['transactions'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash'],
    }
    return jsonify(response), 200
#nodes
@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': 'Our chain was replaced',
            'new_chain': blockchain.chain
        }
    else:
        response = {
            'message': 'Our chain is authoritative',
            'chain': blockchain.chain
        }

    return jsonify(response), 200
```

Otras funciones del sitio

Rutas alternativas

Tenemos rutas alternativas para mejorar la interfaz de usuario del sitio en vez de recuperar datos puros. Las rutas alternativas se usan para las interacciones user-sitio pero no sitio-sitio o nodemcu-sitio.

Estas son /mine/u, /transactions/new/u, chain/u, nodes/register/u

Base de datos (DB)

Las tablas de la base de datos se definen usando la librería SQLAlchemy, en el archivo models/db.py.

Al crear una base de datos, se tiene que configurar el archivo createdb.py que crea las tablas de la db al ser ejecutado. Luego hay que poner el url de nuestra DB en node.py para poder poner nuevos bloques en la db y recuperar la blockchain de esta al reiniciar la app.

```
from flask import Flask, jsonify, request
from flask_sqlalchemy import SQLAlchemy

#DATABASE
db = SQLAlchemy() #our database

#TABLES
class Block(db.Model):
    __tablename__= 'block'
    index = db.Column(db.Integer, primary_key=True)
    timestamp = db.Column(db.String(200))
    proof = db.Column(db.Integer)
```

```
    previous_hash = db.Column(db.String(200))
    transactions = db.relationship("Transaction")

    def __init__(self, index, timestamp, proof, previous_hash):
        self.index = index
        self.timestamp = timestamp
        self.proof = proof
        self.previous_hash = previous_hash

class Transaction(db.Model):
    __tablename__= 'transaction'
    id = db.Column(db.Integer, primary_key=True)
    sender = db.Column(db.String(200))
    recipient = db.Column(db.String(200))
    amount = db.Column(db.Integer)
    cardkey = db.Column(db.String(200))
    location = db.Column(db.String(200))
    date = db.Column(db.String(200))
    index = db.Column(db.Integer, db.ForeignKey('block.index'))

    def __init__(self, sender, recipient, amount, cardkey, location,
date, index):
        self.sender = sender
        self.recipient = recipient
        self.amount = amount
        self.cardkey = cardkey
        self.location = location
        self.date = date
        self.index = index
```

Interfaz de usuario con bootstrap

Se utiliza bootstrap para estilizar el html. Jinja2 para manejar el html con python. Tenemos un archivo base.html que contiene nuestra barra de navegación. El resto de los archivos se agregan a base.html.

```
<!doctype html>
<html lang="en" class="h-100">
      <head>
            <meta charset="utf-8">
            <meta name="viewport" content="width=device-width,
initial-scale=1, shrink-to-fit=no">
            <meta name="description" content="">
            <meta name="author" content="Mark Otto, Jacob Thornton, and
Bootstrap contributors">
```

```
            <meta name="generator" content="Jekyll v4.1.1">
            <title>{% block title %}{% endblock %} Blockchain</title>

            <link rel="canonical"
href="https://getbootstrap.com/docs/4.5/examples/sticky-footer-navbar/">
            <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.2.6/jquery.min.js"><
/script>
            <script>
                  window.jQuery || document.write('<script
src="assets/dist/js/jquery.js"><\/script>');
                  </script>
                  <!-- Bootstrap core CSS -->
                  <!-- <link href="../assets/dist/css/bootstrap.min.css"
rel="stylesheet"> -->
                  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap-t
heme.min.css"
integrity="sha384-6pzBo3FDv/PJ8r2KRkGHifhEocL+1X2rVCTTkUfGk7/0pbek5mMa1u
pzvWbrUbOZ" crossorigin="anonymous">
                  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.m
in.css"
integrity="sha384-HSMxcRTRxnN+Bdg0JdbxYKrThecOKuH5zCYotlSAcp1+c8xmyTe9GY
g1l9a69psu" crossorigin="anonymous">
                  <style>
.bd-placeholder-img {
      font-size: 1.125rem;
      text-anchor: middle;
      -webkit-user-select: none;
      -moz-user-select: none;
      -ms-user-select: none;
      user-select: none;
}

                                                            @media
(min-width: 768px) {

.bd-placeholder-img-lg {

      font-size: 3.5rem;
                                                                  }
                                                            }
                  </style>
                  <!-- Custom styles for this template -->
                  <link href="sticky-footer-navbar.css"
```

```
rel="stylesheet">
                  <script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>
                  {% block style %}
                  {% endblock style %}
      </head>

      <div class="d-flex flex-column h-100">
            <header>
                  <!-- Fixed navbar -->
                  <nav class="navbar navbar-default">
                        <div class="container-fluid">
                              <!-- Brand and toggle get grouped for
better mobile display -->
                              <div class="navbar-header">
                                    <button type="button"
class="navbar-toggle collapsed" data-toggle="collapse"
data-target="#bs-example-navbar-collapse-1" aria-expanded="false">
                                          <span class="sr-only">Toggle
navigation</span>
                                          <span class="icon-bar"></span>
                                          <span class="icon-bar"></span>
                                          <span class="icon-bar"></span>
                                    </button>
                                    <a class="navbar-brand"
href="#">BLOCKCHAIN NODE</a>
                              </div>

                              <!-- Collect the nav links, forms, and
other content for toggling -->
                              <div class="collapse navbar-collapse"
id="bs-example-navbar-collapse-1">
                                    <ul class="nav navbar-nav">
                                          <li class="active"><a href="/">
Home <span class="sr-only">(current)</span></a></li>
                                          <li><a href="#">#</a></li>
                                    </ul>
                                    <p class="navbar-text">Nodo: {{
node_identifier }}</p>
                              </div><!-- /.navbar-collapse -->
                        </div><!-- /.container-fluid -->
                  </nav>
                  <script type="text/javascript">
                        $(window).load(function(){
                              var lpm = {{values|safe}};
                              json_data = JSON.stringify(lpm);
```

```
                                $(".modal-body").text(json_data);
                                $(".modal-title").append({{index}});
                                $('#exampleModal').modal('show');

                            });
                        </script>
                </header>
                <script
src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew
+OrCXaRkfj" crossorigin="anonymous"></script>
                <script>window.jQuery || document.write('<script
src="../assets/js/vendor/jquery.slim.min.js"><\/script>')</script><scrip
t
src="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min
.js"
integrity="sha384-aJ21OjlMXNL5UyIl/XNwTMqvzeRMZH2w8c5cRVpzpU8Y5bApTppSuU
khZXN0VxHd" crossorigin="anonymous"></script>
        </div>

        <div>
                {% block header %}
                {% endblock header %}

                {% block content %}
                {% endblock %}
        </div>
        <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew
+OrCXaRkfj" crossorigin="anonymous"></script>
        <script>window.jQuery || document.write('<script
src="../assets/js/vendor/jquery.slim.min.js"><\/script>')</script><scrip
t
src="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min
.js"
integrity="sha384-aJ21OjlMXNL5UyIl/XNwTMqvzeRMZH2w8c5cRVpzpU8Y5bApTppSuU
khZXN0VxHd" crossorigin="anonymous"></script>
</html>
```

Bitacora: https://github.com/tpII/2020-ps-6-blockchain/wiki/Bitacora

Codigo fuente: https://github.com/tpII/2020-ps-6-blockchain/tree/main

4. Guia de instalacion

Instalación Nodemcu

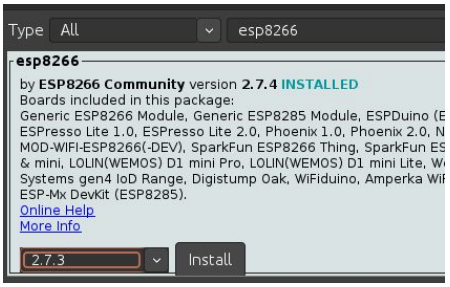
Prerrequisitos

- Arduino 1.8.13 del Sitio Arduino
- Connexion Internet
- Drivers en CH341SER video para windows: https://www.youtube.com/watch?v=JmQbNyUqtCs
- Descargar el archivo nodemcu.ino en el repositorio tpii:https://github.com/tpII/2020-ps-6-blockchain/tree/main
  El archivo se encuentra en 2020-ps-6-blockchain/_ps6files/nodemcu/

Instrucciones

- Iniciar Arduino e ir a Preferences.
- Poner: https://arduino.esp8266.com/stable/package_esp8266com_index.json en el campo Additional Board Manager. Si ya tiene una url, separarlas con commas.

- Abrir Boards Manager en Tools > Board menu y buscar: esp8266.



- Seleccionar la ultima versión e instalar
- Seleccionar ESP8266 board en Tools > Board menu despues de instalar.
- Abrir el código nodemcu.ino
- Cambiar los settings al principio del archivo

```
18
19  //SETTINGS
20  //WIFI SETTINGS
21  String ssid     = "WIFINAME"; //wifi 2.4ghz config   .SETTINGS.
22  String password = "PASSWORD"; //default wifi AP
23
24  //NODE SETTINGS
25  String nodeurl = "http://MYAPP.herokuapp.com/"; //http only, should be best node for less lag
26  String recipientAddress = "MYNODE_IDENTIFIER"; //default recipient uuid, can be any node
27
28  //NODEMCU SETTINGS
29  const String location="MYCITY";
30
```

Ejecución

- Conectar el miniusb del nodemcu a la computadora
- Elegir el puerto en Tools

- En arduino IDE, apretar upload

- Esperar que se envie el codigo a la placa
- Ahora podemos desconectar el nodemcu de la computadora y alimentarlo con otro enchufe 5v (con un cargador de celular anda).
- Podemos controlar el nodemcu por sitio web, buscar el ip del nodemcu en la configuración del router wifi conectado. Si no aparece reconectar el nodemcu.

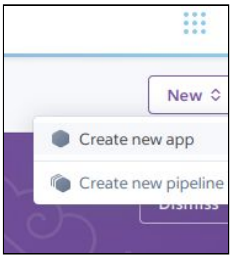
Instalación Nodo Heroku

Prerrequisitos

- Python 3.9.0 https://www.python.org/downloads/
- Connexion Internet
- Cuenta heroku https://www.heroku.com/
- Descargar el repositorio tpii:https://github.com/tpII/2020-ps-6-blockchain/tree/main
- Git https://git-scm.com/downloads
- Opcional: visualizador JSON https://chrome.google.com/webstore/detail/json-viewer/gbmdgpbipfallnflgajpaliibnhdgobh

Instrucciones

- Crear una aplicación en heroku



- Ir a open app para recuperar la url de nuestro nodo

- Poner los archivos del repositorio en una carpeta y abrir la termina allí
- Cambiar el host (*myhost)* a la url de nuestro nodo heroku

```
myport = 80
myhost = "MYAPP.herokuapp.com"
```

- Ir a deploy y seguir los comandos para iniciar el sitio:

```
heroku login
heroku git:clone -a MYAPP
cd MYAPPFOLDER
git add .
git commit -am "make it better"
git push heroku master
heroku config:set WEB_CONCURRENCY=1
```

Ejecución

- Abrir la terminal en la carpeta creada
- En la terminal ejecutar el comando: Heroku open

Para agregar nodos crear una nueva app y hacer lo mismo, no hay que olvidarse de cambiar *myhost* para cada nodo. Luego cada nodo tendría que registrar la url de sus nodos vecinos en nodes/register.

Instalación Base de datos

Prerrequisitos

- Python 3.9.0 https://www.python.org/downloads/
- Connexion Internet
- Cuenta heroku https://www.heroku.com/
- PostgreSQL https://www.postgresql.org/download/
- Nodo heroku con la blockchain
- Pip (o pip3) install todo el contenido de requirements.txt

Instrucciones

- En el directorio de nuestro repo nodo hacemos el commando

```
heroku addons:create heroku-postgresql:hobby-dev --app MYAPP
```

- Luego recuperar la URL de nuestra DB con

```
heroku config --app MYAPP
```

- Esto devolverá la DATABASE_URL que necesitamos, copiamos esta URL
- Abrir el archivo createdb.py y poner esta url en SQLALCHEMY_DATABASE_URI

```python
from models.db import *
from flask import Flask, jsonify, request, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'MYAPP DATABASE_URL ACA'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db.init_app(app)

with app.app_context():
    db.create_all()
```

- Ahora ejecutar: `python createdb.py` para crear las tablas de nuestra DB
- En `node.py` cambiar el URL de la misma manera

```python
ENV = 'prod'   #change to dev for development, change to prod to deploy
if ENV == 'dev':
    app.debug = True
    app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://admin:admin@localhost/ps6db' #db config p://user:password@url/dbname
else:
    app.debug = False
    app.config['SQLALCHEMY_DATABASE_URI'] = 'MYAPP DATABASE_URL'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db.init_app(app) #init db
```

- Ahora nuestra base de datos postgres heroku está instalada y conectada al nodo

Ejecución

- Para acceder a nuestra DB ejecutamos este comando:

```
heroku pg:psql --app MYAPP
```

- Ahora podemos verificar las tablas de nuestra DB con el comando `\dt`
- Para ver los bloques: `SELECT * FROM block;`

Ahora al iniciar la aplicación heroku se creará un bloque génesis y se cargará a la base de datos. Podemos cargar bloques en la DB con /mine. Cuando reiniciamos la aplicación recuperar la blockchain guardada en la base de datos.

Para borrar todos los datos si queremos rehacer la blockchain:

```
DELETE FROM transaction;
DELETE FROM block;
```

5. Problemas y Soluciones

Repositorios

Hubieron varios problemas con relación al uso del repositorio github suministrado por la catedra e integraciones con aplicaciones de terceros. En nuestro caso particular, gitkraken y heroku. Uno de los integrantes del grupo usa gitkraken para administrar su repositorio, pero dado a que el mismo estaba dentro de la organización TPII, se necesitaba solicitar permiso al administrador del grupo para poder utilizar la herramienta. La solución a esto fue utilizar la CLI de git clásica, y realizar todos los commits a mano.

Lo mismo sucedió con heroku. Heroku tiene una integración la cual permite hacer un push automático a github de las cosas pusheadas a los servidores de heroku, sin tener que preocuparse que algún elemento esté desactualizado. En nuestro caso, tratar de conectar ambas herramientas resultó en la misma conducta que el primer punto. Nuestra solución fue separar a heroku y github en dos repositorios distintos. Todos los cambios hechos en heroku van a ser subidos a github cuando se hayan probado que funcionen.

Ahora tenemos 3 repositorios: el de heroku(heroku), el de la cátedra(tpii) y el de desarrollo (origin), la branch principal es: main.

Valor de concurrencia en Heroku

Al realizar el deploy de la página ps6node, nos dimos cuenta que estábamos encontrando que la pagina tenia dos blockchains con dos id de nodos distintas, a pesar de tener un solo dyno configurado en la página de heroku. Este problema fue solucionado limitando la variable WEB_CONCURRENCY a 1 en la consola de heroku.

Configuración del url nodo en nodemcu

Si queremos cambiar las variables de: recipiente de la transacción de los datos y sitio web que lleva los datos, se puede utilizar el comando *sendto* durante la ejecución del nodemcu. Hay que pasar estos argumentos desde el webserver y recuperarlos. No terminamos de hacer la recuperación de los argumentos por http, se quería usar una función string splitter pero no anduvo. Sin embargo, no sería muy complicado terminar la implementación dividiendo el comando en dos para cambiar cada variable por separado.

Problema de fecha y de dirección de los participantes

Cuando hace una transacción, el nodemcu devuelve una fecha equivocada. Esto se debe a que se necesita tener que configurar la fecha y tiempo actual en la placa. Además, se podría mejorar las direcciones de los nodos y placas para que sean estandarizadas con longitud y formato fijo.

Optimizar la búsqueda de transacciones

En vez de una lista enlazada se podría poner las transacciones en un árbol en cada bloque, la criptomoneda bitcoin utiliza un merkle tree para guardar las transacciones en bloques.

Documentacion doxygen

Se puede utilizar la herramiento doxygen pero mejorar la documentación del proyecto y generar un doc html de todo el código fuente, el código nodemcu tiene todas la funciones comentadas para el formato javadoc para doxygen, no se genero el documento pero se podría hacer si crece el proyecto.

6. Documentación en Formato Gráfico y Video

Sistema físico

Foto del sistema, con el lector RFID conectado al nodemcu:

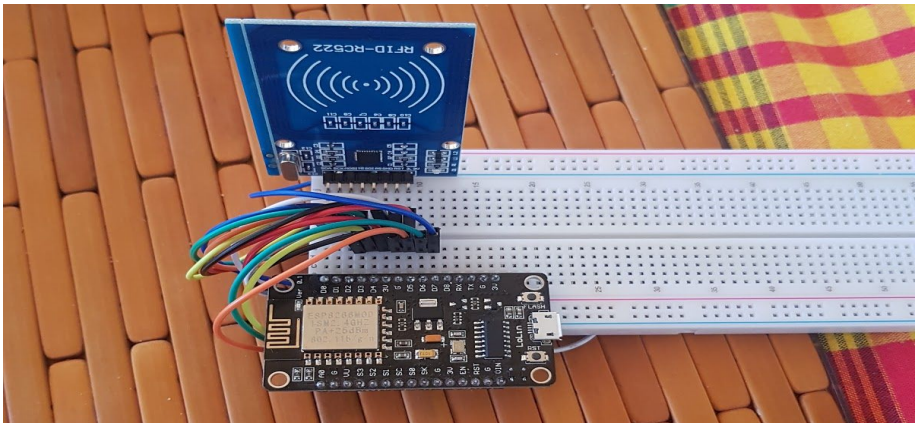


Fig.6: foto nodemcu y scanner

Video funcionamiento

Video del nodemcu

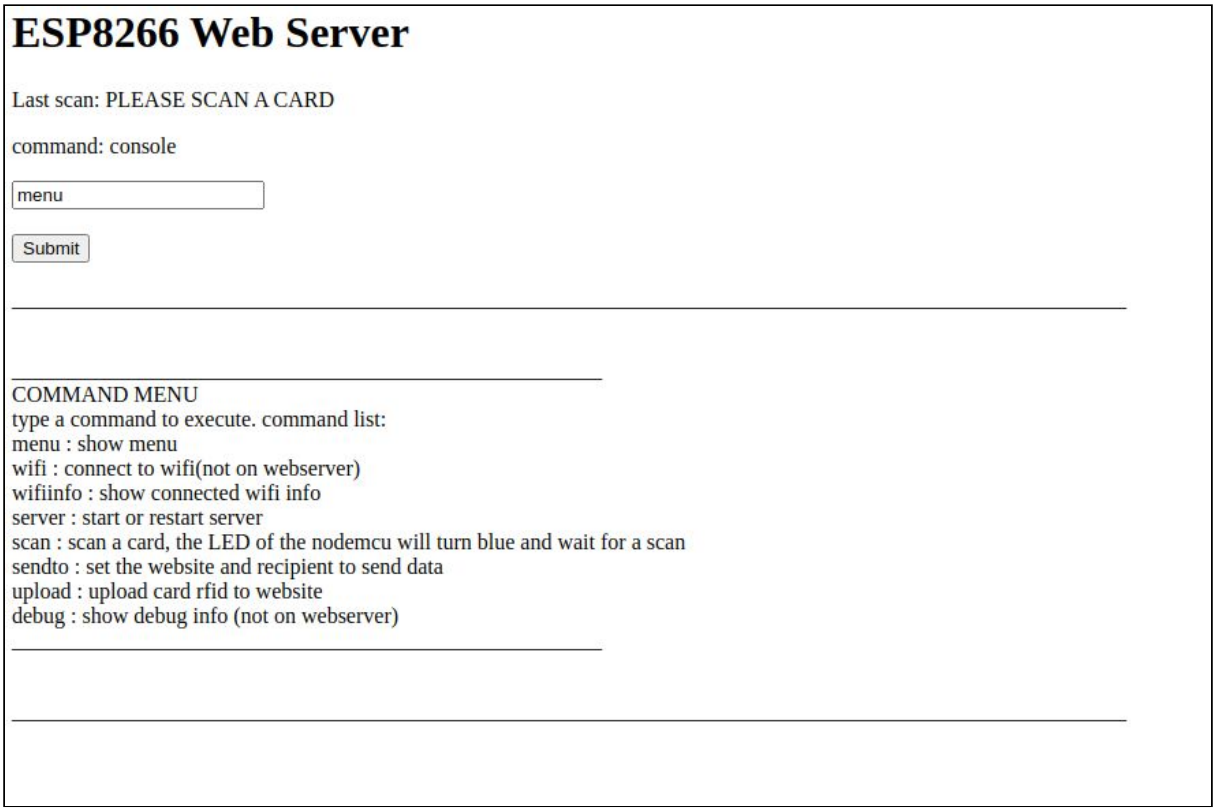
https://drive.google.com/file/d/1hjdNiYHDpQqeSX6rVjZbNy5qa4uWZB5H/view?usp=sharing

Video del nodo heroku

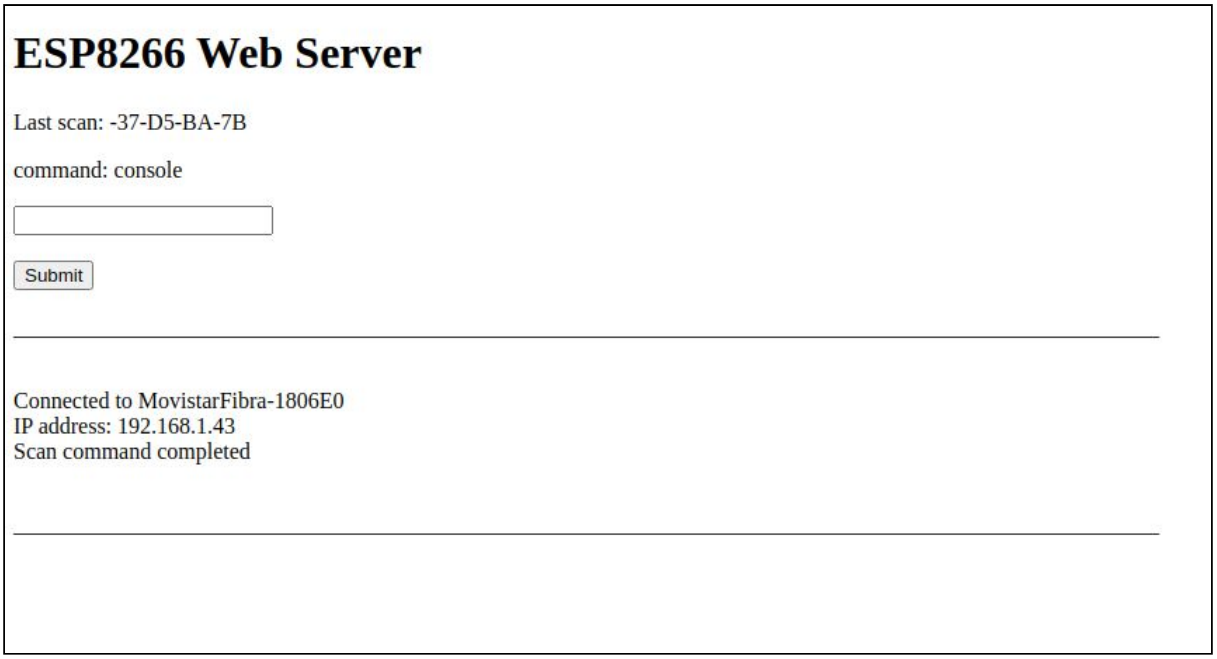
https://drive.google.com/file/d/1cuAL_J52lIex-LUk7_5xYwPHuI1wCJ7Z/view?usp=sharing

NodeMcu WebServer

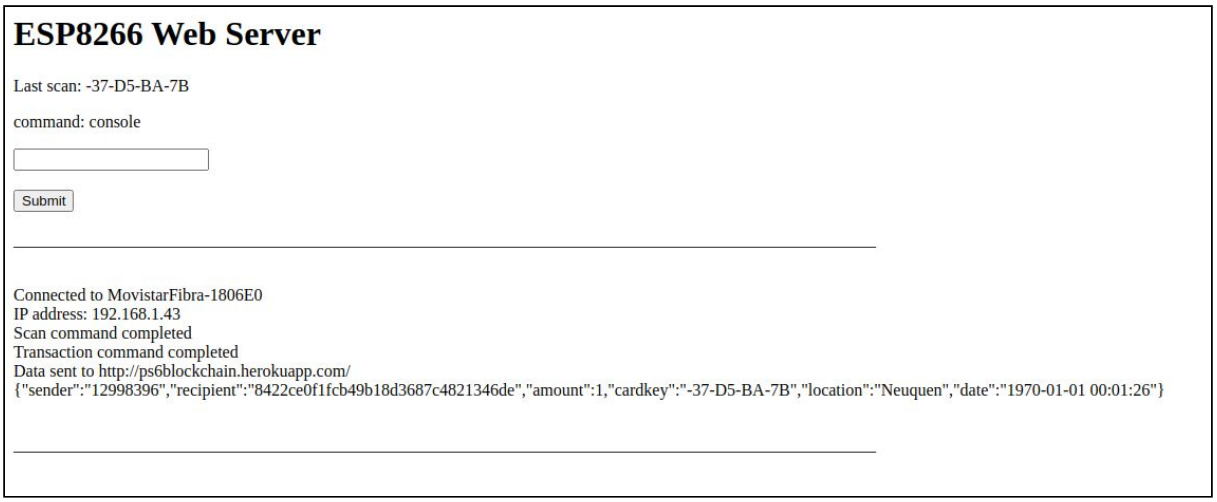
Inicio y comando menú



Commando scan

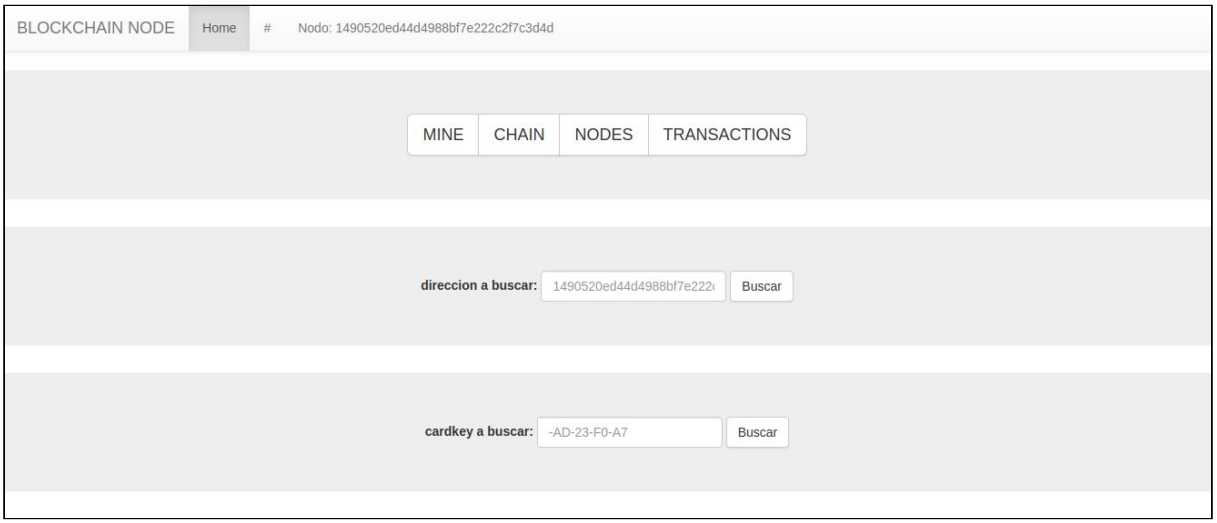


Commando upload



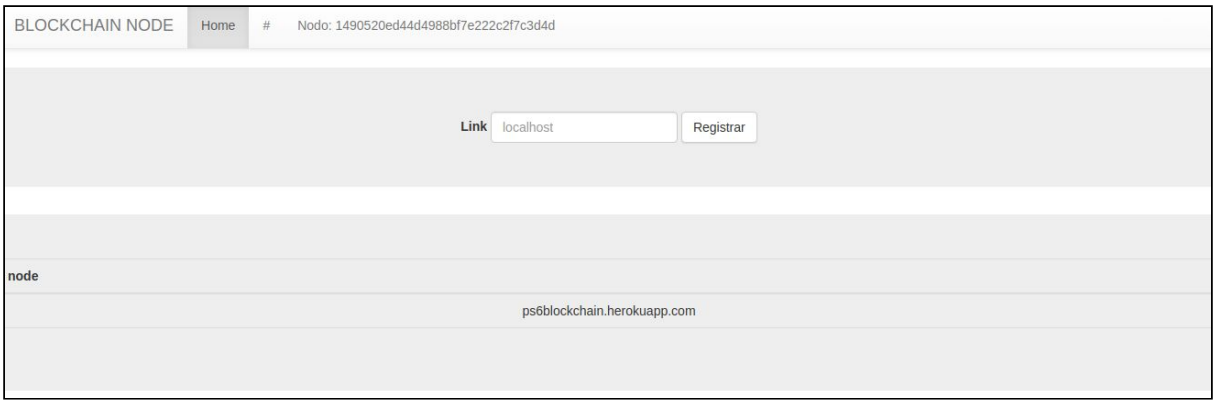
Heroku Node

Pagina principal

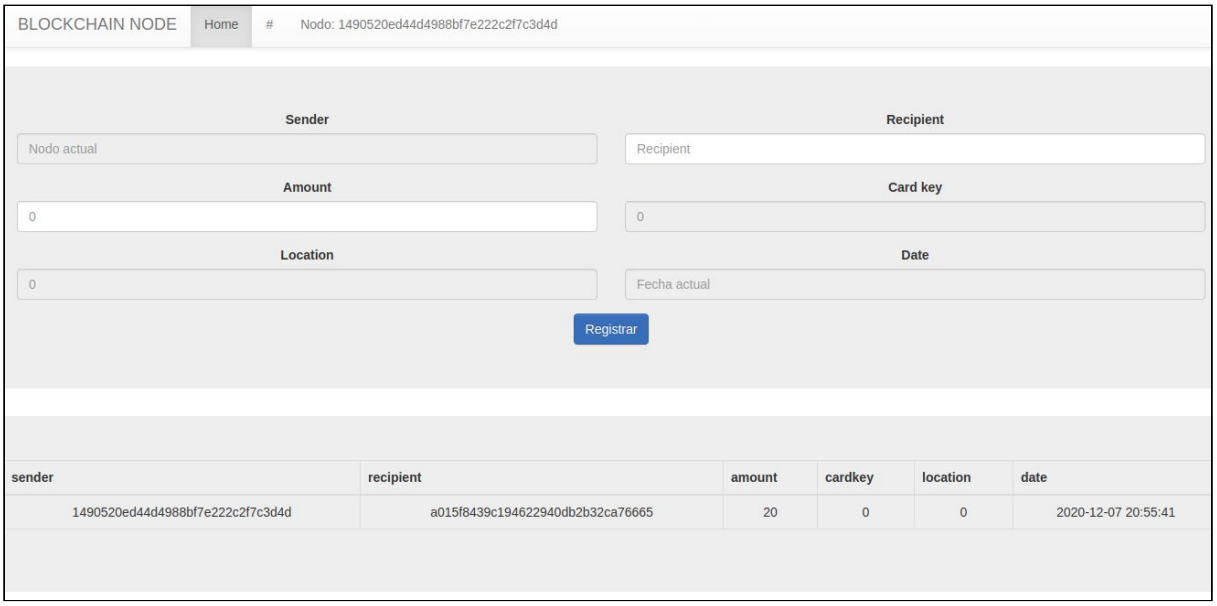


Página chain

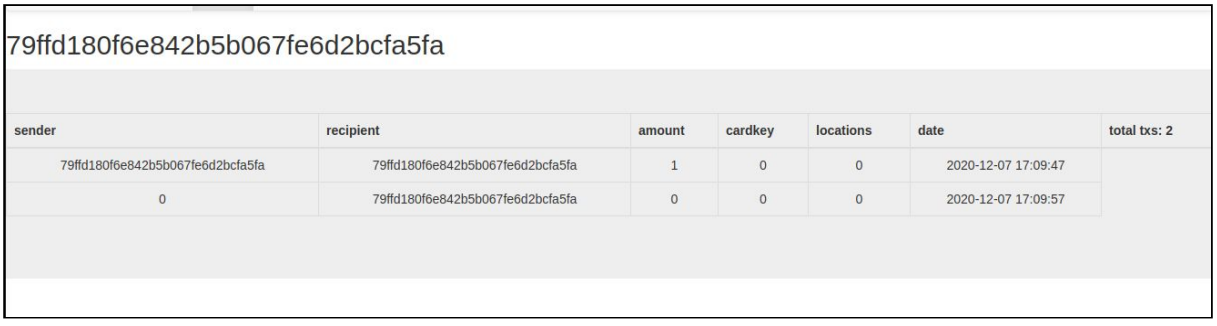
Página para agregar nodos



Crear transacciones



Busqueda



7. Conclusiones

El proyecto nos ha aportado conocimientos sobre el desarrollo en placas estándares arduino y nodemcu. Nos permite descubrir librerías útiles y varios módulos del nodemcu, y nos ha permitido aprender acerca de distintos métodos de comunicación. Además, se ha podido practicar como se conecta hardware físicamente.

Se pone en práctica los protocolos de comunicaciones aprendidos como http para realizar operaciones GET o POST, formatos de datos JSON para poder procesar la información y realizar la comunicación entre placa-sitio y sitio-sitio.

Se ha utilizado la bitácora para documentar el desarrollo diario de las aplicaciones, incluyendo avances realizados en la investigación de la tecnología en sí, herramientas de documentación como doxygen, uso de heroku para implementación de múltiples sitios y colaboración múltiple en la creación de los mismos y la utilización de herramientas de cooperación y comunicación entre personas como discord y slack para poder discutir dudas y avances entre alumno y alumno o entre alumno y profesor.

Se han podido utilizar nuevas tecnologías para los integrantes en la realización de la página web, como ha sido django para la integración python-html, flask para la creacion de paginas web con python, y la utilización de postgresql para el mantenimiento de la consistencia de la base de datos de la página web en cada uno de los nodos.