

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ
IFCE

Francisco Iuri Gomes Mendes

Fortaleza, 2020

Francisco Iuri Gomes Mendes

Relatório sobre o projeto de pesquisa “Ambiente virtual para o ensino de rastreador econômico utilizando o microcontrolador ESP8266”

Relatório técnico informativo sobre o que foi feito na pesquisa e os métodos utilizados para a concepção do rastreador.

Fortaleza, 2020

RESUMO

Este relatório tem como objetivo apresentar as tecnologias usadas, seus algoritmos e como implementar um rastreador da forma mais econômica possível utilizando um Microcontrolador ESP8266 e exibir a localização do dispositivo em uma página Web apresentando o local em um Mapa do Google Maps.

SUMÁRIO

1. INTRODUÇÃO	5
2. CONFIGURAÇÕES NECESSÁRIAS	7
3. CÓDIGO ARDUINO	9
4. CÓDIGOS JAVASCRIPT	13
5. IMPLANTAR NO HEROKU	17
6. CONCLUSÃO	19
7. REFERÊNCIAS	20

1. INTRODUÇÃO

Para a criação de um rastreador em tese é necessário um dispositivo que contenha um módulo GPS para a obtenção da latitude e longitude, ou seja, das coordenadas que são usadas para indicar o local de qualquer lugar na Terra.

Mas foi possível obter a localização exata somente utilizando um microcontrolador baseado em ESP8266 usando a biblioteca do Arduino *WiFi Location*. A biblioteca se comunica com duas API's do Google Maps para a obtenção da localização, como será feito isso, será explicado mais adiante.

Portanto, o único hardware necessário foi um microcontrolador baseado em ESP8266, o utilizado é uma Wemos D1 que possui um processador ESP8266-12E, WiFi nativo padrão, pode ser alimentado em 5 volts em uma porta Micro USB, dessa forma, basta conectar a uma porta USB para receber alimentação e ficar ligada, também pode ser programada utilizando a IDE do Arduino.

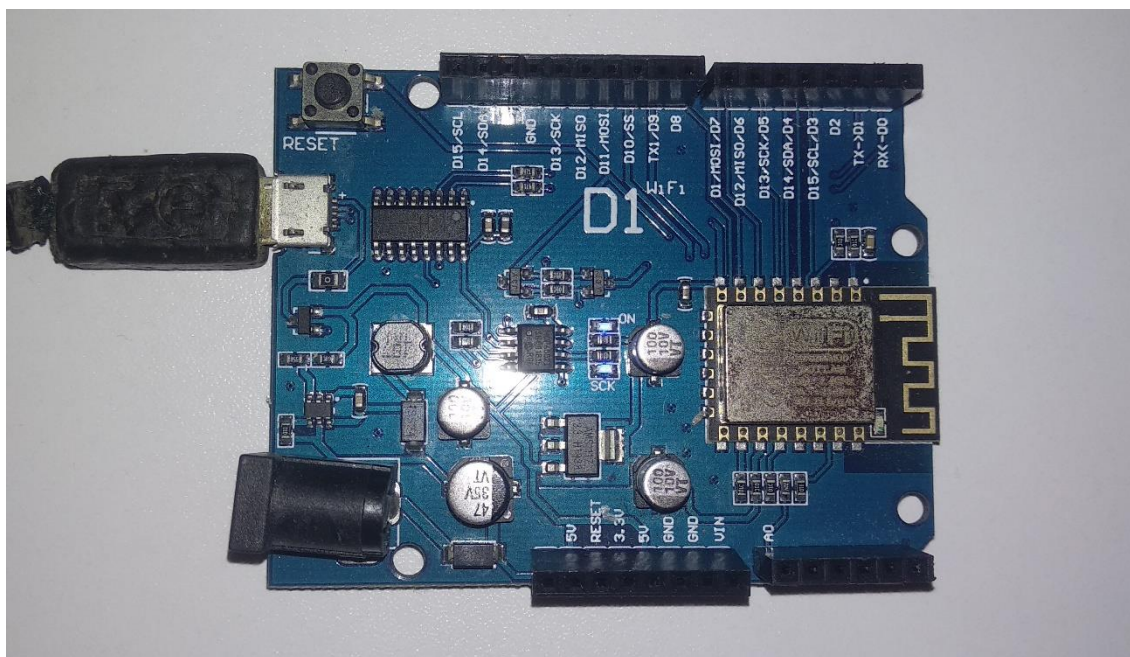


Figura 1 - Wemos D1 conectada a uma porta USB

Para a implementação do código na placa Wemos D1 e a exibição em uma página Web será usado duas linguagens de programação, C e JavaScript. A linguagem C será usada para o carregamento do código no microcontrolador, já a linguagem JavaScript será usada para a exibição da página Web. Assim como é preciso usar a linguagem de marcação HTML e CSS para o estilo da página.

Outra tecnologia utilizada é um Banco de Dados (BD) em tempo real, o usado é o Firebase, BD do tipo Chave/Valor bastante usado para projetos *IoT* (Internet das Coisas). A placa envia os dados (latitude, longitude, última atualização) para o Firebase e o código em JavaScript trata de carregar esses dados para a exibição na Web, o

Firebase se sincroniza com cada dispositivo conectado a ele e trata de atualizar automaticamente os dados.

Já para não ser necessário gastar com hospedagem e poder verificar a localização do dispositivo em qualquer lugar será usado o Heroku, que é uma plataforma em nuvem que possibilita armazenar o projeto de forma gratuita.

2. CONFIGURAÇÕES NECESSÁRIAS

O primeiro passo necessário para a implementação dos códigos é a configuração das API's no site para developers do Google Maps e do Firebase.

São necessárias duas API's, a *Geolocation API* e *Maps JavaScript API*, a *Geolocation* é necessária para a biblioteca WiFi Location do Arduino conseguir saber as coordenadas exatas de onde está o dispositivo, assim como a Maps JavaScript, mas ela também é necessária para a exibição na página Web, sendo assim é necessário uma chave de acesso fornecida pela API para colocar tanto no código do Arduino como do JavaScript. É fundamental também a inclusão de dados de cartão de crédito na API, pois houve uma mudança de cobrança no faturamento da API, mas ela funciona de forma gratuita se não ultrapassar os 200 dólares por mês, o preço é calculado de acordo com as solicitações aos servidores do Google, mas isso não é preocupante pois é permitido fazer milhares de solicitações de forma gratuita.

O próximo passo é configurar o Firebase, basta criar normalmente um projeto no site do mesmo e inserir os valores do tipo Chave/Valor para o armazenamento dos dados. A propriedade da tabela fica sendo:

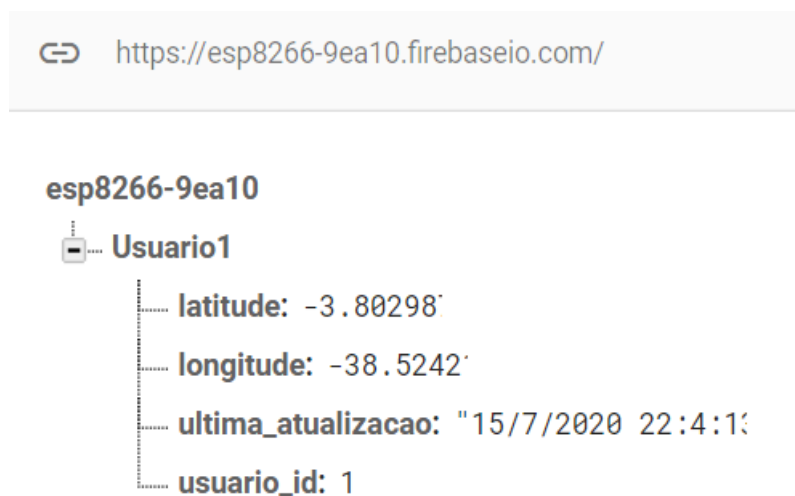


Figura 2 - Configuração BD Firebase

O link *esp8266-9ea10.firebaseio.com* é necessário para identificar o projeto para a inserção e leitura de dados. O campo *esp8266-9ea10* mostrado é o nome do projeto e seu ID informado pelo Firebase. As chaves *latitude* e *longitude* são para armazenar os valores da latitude e longitude. A chave *ultima_atualizacao* é para saber o horário e a data da última vez que o dispositivo enviou informações de sua posição para o servidor do BD. A chave *usuario_id* serve para diferenciar caso seja necessário criar e exibir outro dispositivo. Os valores na imagem são os de onde o dispositivo se encontrava no momento da captura, sua última atualização e o do usuário a ser mostrado.

É necessário mudar as regras de autenticação de escrita e leitura para verdadeiro, com a finalidade de o Arduino e o JavaScript poderem acessar os dados com facilidade.

```
{  
  "rules": {  
    ".read": true,  
    ".write": true  
  }  
}
```

Figura 3 - Regras Firebase

Agora é preciso acessar as configurações do Firebase para a obtenção das chaves de autenticação para usar no Arduino e no JavaScript.

```
const firebaseConfig = {  
  apiKey: [REDACTED],  
  authDomain: [REDACTED],  
  databaseURL: [REDACTED],  
  projectId: [REDACTED],  
  storageBucket: [REDACTED],  
  messagingSenderId: [REDACTED],  
  appId: [REDACTED],  
  measurementId: [REDACTED]  
};
```

Figura 4 - Objeto contendo as chaves de autenticações do Firebase

No Arduino é necessário somente a *apiKey* e a *authDomain*, as outras serão necessárias no JavaScript.

3. CÓDIGO ARDUINO

No código no arduino é necessário a inclusão de algumas bibliotecas, a *ESP8266WiFi* que serve para fazer funcionar a placa na IDE do Arduino, a *WiFiLocation* que é a forma para obter a localização sem usar um módulo GPS, a biblioteca envia uma solicitação à API *Geolocation* do Google Maps com uma lista de todos os pontos de acesso WiFi que a placa está escutando.

```
[
{"macAddress": "XXXXXXXXXX", "signalStrength": -74, "channel": 1},
{"macAddress": "XXXXXXXXXX", "signalStrength": -58, "channel": 3},
{"macAddress": "XXXXXXXXXX", "signalStrength": -66, "channel": 3},
{"macAddress": "XXXXXXXXXX", "signalStrength": -91, "channel": 4},
{"macAddress": "XXXXXXXXXX", "signalStrength": -78, "channel": 9},
{"macAddress": "XXXXXXXXXX", "signalStrength": -94, "channel": 11},
{"macAddress": "XXXXXXXXXX", "signalStrength": -84, "channel": 11}]
```

Figura 5 – Solicitação enviada a API do Google Maps

A API responde com a localização correta.

```
Latitude: -3.8030174
Longitude: -38.5241585
Accuracy: 21
```

Figura 6 - Retorno da localização pela API

Outra biblioteca necessária é a *time* que será para saber o horário correto da última atualização informada pelo dispositivo. Assim como é preciso a *FirebaseArduino* para fazer a inserção dos dados no Banco de Dados.

```
1//Início das bibliotecas utilizadas pelo WiFi Location
2#ifdef ARDUINO_ARCH_SAMD
3#include <WiFi101.h>
4#elif defined ARDUINO_ARCH_ESP8266
5#include <ESP8266WiFi.h>
6#elif defined ARDUINO_ARCH_ESP32
7#include <WiFi.h>
8#else
9#error Wrong platform
10#endif
11#include <WifiLocation.h>
12//Fim das bibliotecas utilizadas pelo WiFi Location
13#include <time.h> //Biblioteca para a atualização do horário
14#include <FirebaseArduino.h> //Biblioteca para a escrita de dados no BD
```

Figura 7 - Bibliotecas utilizadas no código Arduino

O próximo passo é inserir os dados para conexão, tanto da rede WiFi ao qual a placa vai conectar-se como a chave da API do Google Maps e as do Firebase. Também ocorre a chamada da API pelo construtor da classe *WiFiLocation*, também é definido o fuso horário da localidade para exibir o horário corretamente na variável *timezone*, onde 3600 significa uma hora em segundos e é multiplicado por 3, pois a localidade onde está o dispositivo está no fuso horário GMT -3, já a variável *dst* significa quantas horas são atrasadas ou adicionadas em relação ao horário padrão.

```

16 const char* googleApiKey = "CHAVE DA API"; //inserir API do Google Maps
17 const char* ssid = "NOME DO WIFI"; //Nome da Rede WiFi
18 const char* passwd = "SENHA DO WIFI"; //Senha da Rede WiFi
19
20 #define FIREBASE_HOST "NOME_DO_PROJETO.firebaseio.com" //Host do Firebase
21 #define FIREBASE_AUTH "KEY PARA CONEXÃO" //Key do Firebase
22
23 WiFiLocation location(googleApiKey); //Chave inserida ao construtor da classe WiFiLocation
24
25 int timezone = -3 * -3600; //fuso horário da localidade
26 int dst = 0; //Horário de Verão

```

Figura 8 – Valores para configuração da placa

Na função *setup* é iniciada definindo a taxa de comunicação para 115200 bits por segundo, logo após isso é definido que será utilizado uma placa com um processador ESP8266, depois é chamada a função para conexão da rede WiFi. Após a conexão ser sucedida, é configurado o tempo, em seguida é chamada a função para iniciar o Firebase.

```

28 void setup() {
29   Serial.begin(115200); //Configurando a taxa de comunicação para 115200 bps
30   #ifdef ARDUINO_ARCH_ESP8266 //Configura para usar esp8266
31     WiFi.mode(WIFI_STA);
32   #endif
33   WiFi.begin(ssid, passwd); //Inicia a configuração na rede WiFi
34   while (WiFi.status() != WL_CONNECTED) { //Enquanto a conexão não tiver sucesso
35     Serial.print("Conectando a rede WiFi: ");
36     Serial.println(ssid);
37     Serial.print("Status = ");
38     Serial.println(WiFi.status());
39     delay(500); //Aguarda 0,5 segundos e tenta novamente
40   }
41   configTime(timezone, dst, "pool.ntp.org", "time.nist.gov"); //Função para configurar o tempo
42   Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH); ///Inicia o Firebase
43 }

```

Figura 9 - Setup do código no Arduino

É criada uma função para armazenar os valores a serem enviados ao Firebase, a função recebe como parâmetros a latitude, longitude e o tempo. Caso as coordenadas sejam zero, os valores não são enviados ao servidor, caso sejam valores diferentes de zero, os dados são enviados. Se existe algum erro com o Firebase, é chamada a função de falha do próprio Firebase e exibe a mensagem de erro no Monitor Serial do Arduino. Cada vez que as informações são enviadas, há um atraso de 1 segundo.

```

45 void enviarFirebase(float longi, float lati, String tempo) { //Passado como parâmetro a longitude, latitude e tempo
46   if (longi == 0 && lati == 0) { //Caso receba longitude e latitude zerados não é enviado para o Firebase
47     Serial.println("Não computado");
48   }
49   else { //Caso receba valores verdadeiros, são enviados para o Firebase
50     Firebase.setFloat("Usuariol/longitude", +longi); //Armazenada a longitude no servidor
51     Firebase.setFloat("Usuariol/latitude", +lati); //Armazenada a latitude no servidor
52     Firebase.setString("Usuariol/ultima_atualizacao", String(tempo)); //Armazenada o tempo no servidor
53     if (Firebase.failed()) { //Caso haja algum erro na conexão com o Firebase esta função é ativada
54       Serial.print("Falhou");
55       Serial.println(Firebase.error());
56       return;
57     }
58     delay(1000); //Atraso de 1 segundo a cada envio de dados
59   }
60 }

```

Figura 10 - Função Firebase do Código Arduino

Antes de iniciar a função *loop*, é necessária definir a variável contador, pois será necessária para o reiniciamento da placa, afim de economizar o envio de dados e a utilização dos servidores gratuitos. No início da função *loop* é chamada a função da biblioteca *WiFiLocation* e armazenada na variável *loc*, após isso é criada uma variável e uma struct para o armazenamento do horário, depois a adição de 1 mês a mais na variável *mes* e de 1900 anos a mais na variável *ano*, pois essas informações são recebidas com 1 mês e 1900 anos de atraso respectivamente. Após isso, é mostrado no Monitor Serial as informações da latitude, longitude e tempo. Assim que recolhidas as informações de coordenadas e do horário, elas são enviadas para a função *enviarFirebase*. Para a placa ser reiniciada o contador tem que chegar a 0, como foi definido fora do escopo da função o contador como 5, levará no mínimo 50 segundos para o contador chegar a 0 e a placa ser reiniciada, pois no fim do *loop* o contador é decrementado e há um atraso de 10 segundos a cada execução da função.

```

62 int cont = 5; //Contador definido para reiniciar a placa
63 void loop() {
64     location_t loc = location.getGeoFromWiFi(); //Chamada da função WiFiLocation
65     time_t now = time(nullptr); //Criação de variável para armazenar horário
66     struct tm* p_tm = localtime(&now); //Criação de struct para armazenar horário
67     int mes = p_tm->tm_mon + 1; //Define a variável mês para receber +1
68     int ano = p_tm->tm_year + 1900; //Define a variável ano para receber +1900
69     Serial.println("Solicitando Localização");
70     Serial.println(location.getSurroundingWiFiJson()); //JSON com localização retornado
71     Serial.println("Latitude: " + String(loc.lat, 7)); //Exibe informação da latitude no Monitor Serial
72     Serial.println("Longitude: " + String(loc.lon, 7)); //Exibe informação da longitude no Monitor Serial
73     Serial.println("Precisão: " + String(loc.accuracy)); //Exibe informação da precisão no Monitor Serial
74     float longi = loc.lon; //Variável longi recebe os valores da longitude recebidos
75     float lati = loc.lat; //Variável lati recebe os valores da longitude recebidos
76     String tempo = String(p_tm->tm_mday) + String("/") + String(mes) + String("/") + String(ano) + String(" ") + String(p_tm->tm_hour)
77     + String(":") + String(p_tm->tm_min) + String(":") + String(p_tm->tm_sec);
78     //Variável tempo recebe os valores do horário no padrão DD/MM/AAAA HH:mm:ss
79     Serial.println(tempo); //Exibe informação do horário no Monitor Serial
80     Serial.println(cont); //Exibe valor atual do contador no Monitor Serial
81     enviarFirebase(longi, lati, tempo); //Envia os dados da latitude, longitude e tempo para a função enviarFirebase
82     if (cont == 0) { //Caso o contador chegue a 0 a placa é reiniciada
83         Serial.println("Reiniciar");
84         ESP.restart();
85     }
86     cont--; //Decrementa o contador
87     delay(10000); //Atraso de 10 segundos
88 }

```

Figura 11 - Função Loop do Código Arduino

Por último somente é necessário carregar o código na placa e esperar a exibição com sucesso das mensagens enviadas ao Monitor Serial.

```

Conectando a rede WiFi: ██████████
Status = 6
Solicitando Localização
[
{"macAddress": "██████████", "signalStrength": -51, "channel": 3},
{"macAddress": "██████████", "signalStrength": -49, "channel": 3},
{"macAddress": "██████████", "signalStrength": -82, "channel": 4},
{"macAddress": "██████████", "signalStrength": -87, "channel": 8},
{"macAddress": "██████████", "signalStrength": -89, "channel": 11},
{"macAddress": "██████████", "signalStrength": -87, "channel": 11}]
Latitude: -3.8029888
Longitude: -38.5242386
Precisão: 30
18/7/2020 9:9:26
5

```

Figura 12 - Resultado no Monitor Serial do Código no Arduino

4. CÓDIGOS JAVASCRIPT

O primeiro passo para iniciar a escrita dos códigos em JavaScript é necessário importar alguns módulos. O primeiro é o Express que será necessário para gerenciar as requisições HTTP das URLs, definir a porta a ser usada na conexão e a localização do arquivo a ser renderizado na resposta. O segundo é o EJS que é uma engine de visualização que permite gerar a linguagem de marcação HTML com JavaScript. O último módulo necessário é o Firebase, que será necessário para carregar os dados e exibir na página. Para verificar se os módulos foram adicionados com sucesso, basta acessar o arquivo chamado *package.json*, nele mostrará todas as dependências do projeto, assim como as informações do projeto e as variáveis utilizadas para debug.

No início dos códigos no JS é necessário criar um arquivo para importar os módulos já mencionados. Informar que será utilizado como motor de visualização o EJS e o local onde estará as *views*. É necessário aplicar que será utilizado códigos estáticos para carregar imagens, CSS e outros códigos JavaScript. Após isso, é aplicado a porta que será utilizada para receber as solicitações enviadas pelos códigos, no caso, a porta 3000. Por último, é apresentado o caminho a qual a página será renderizada e o arquivo a ser exibido.

```
JS app.js > ...
1 | const express = require("express") //Importa o módulo Express
2 | const app = express() //Inicia o Express
3 | const firebase = require("firebase") //Importa o Firebase
4 |
5 | app.set("view engine", "ejs") //Informa que o tipo de view será no formato EJS
6 | app.set("views", "./resources/views") //Informa o local de onde estará as views
7 |
8 | app.use(express.static(__dirname + '/resources')) //Informa que os arquivos nesta pasta serão estáticos
9 | app.listen(process.env.PORT || 3000); //Porta a ser usada
10 |
11 | app.get("/", function(req, res) { //Renderiza no Index(/) o arquivo informado
12 |   res.render("index.ejs")
13 | })
```

Figura 13 - Arquivo inicial Código JavaScript

Na pasta *resources*, onde estão localizados os arquivos estáticos, há três subpastas, a pasta *css*, responsável por conter o arquivo *css* de estilo da página, a *images* que contém o ícone da página e a *js* que há dois arquivos responsáveis pela geração do mapa. Nesta pasta também está a *view* que mostrará a página web.

O primeiro código trata-se da exibição na página, é criada uma função para receber os parâmetros vindo do Firebase (latitude, longitude e data), adicionados a um objeto que trata de centralizar o mapa nessa localidade e adiciona um zoom a página, nesse caso 14. Após isso é tratado de exibir na página. Para ter a data e o horário informado somente é possível através de um marcador que também será mostrado no local exato onde está o dispositivo. É exibido a informação do horário ao passar o mouse por cima. Por último é necessário informar a chave de acesso que foi solicitada através da API do Google Maps e passar como parâmetro para exibição.

```
resources > js > JS Mapjs > ...
1  iniciarMap = function(lati, longi, data_atualizacao) { //Parâmetros recebidos pela função para exibir na página: Latitude, Longitude e Data
2    var local = { lat: lati, lng: longi } //Objeto recebe latidude e longitude para exibição
3    var opcoes = {
4      center: local,
5      zoom: 14 //Adiciona zoom na exibição
6    }
7
8    map = new google.maps.Map(document.getElementById('map'), opcoes) //Exibe na página
9    var marker = new google.maps.Marker({ //Adiciona um marcador no local exato da localização e informa a data
10     position: local, //Posição do marcador
11     map: map,
12     title: data_atualizacao //Título
13   })
14 }
15
16 const googleMapsScript = document.createElement('script'); //configuração de key de acesso
17 googleMapsScript.src = "https://maps.googleapis.com/maps/api/js?key=_____&callback=iniciarMap";
18 document.head.appendChild(googleMapsScript);
```

Figura 14 - Map no Código JavaScript

O segundo e último código é o *MapaDAO* que é responsável por trazer as informações do Firebase, o código carrega os resultados enviados pela placa ao servidor. No início do código é recebido os parâmetros de configuração de acesso ao BD, depois é informado a tabela a qual será utilizado e adicionado as colunas as variáveis criadas. Por último, é enviado os dados a função *iniciarMap* que exibirá na página.

```
resources > js > JS MapaDAOjs > ...
1  // Inicialização Firebase
2  firebase.initializeApp({
3    apiKey: "_____",
4    authDomain: "_____",
5    databaseURL: "_____",
6    projectId: "_____",
7    storageBucket: "_____",
8    messagingSenderId: "_____",
9    appId: "_____",
10   measurementId: "_____"
11 });
12 const analytics = firebase.analytics();
13
14 var referencia = firebase.database().ref("Usuario1") //Tabela a qual será utilizada
15 referencia.on('value', function(dataSnapshot) {
16   var lati = dataSnapshot.val().latitude //Carregando valor da latitude
17   var longi = dataSnapshot.val().longitude //Carregando valor da longitude
18   var pegar_data = dataSnapshot.val().ultima_atualizacao //Carregando valor da data
19   iniciarMap(parseFloat(lati), parseFloat(longi), pegar_data) //Enviando parâmetros a função responsável por mostrar o mapa
20 }
21 })
```

Figura 15 - MapaDAO no Código JavaScript

No arquivo CSS é informado que o estilo da página terá altura de 100%, terá margem 0 em todos os lados e terá também distância 0 entre o conteúdo e suas bordas.

```
resources > css > # Mapa.css > html
1  html,
2  body {
3      height: 100%;
4      margin: 0;
5      padding: 0;
6  }
7
8  #map {
9      height: 100%;
10 }
11
12 #corpoMapa {
13     height: 100%;
14 }
```

Figura 16 - CSS no Código JavaScript

Na pasta *images*, há somente um arquivo que trata-se do ícone da página que serpa uma foto da Wemos D1.

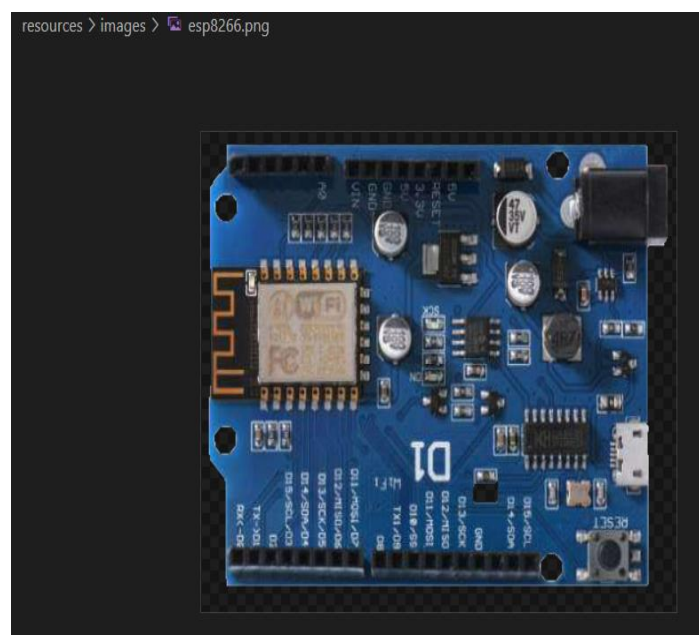


Figura 17 - Ícone da página Web

Na pasta *views*, há o arquivo *EJS* responsável por mostrar a página Web. A página possui o título “Mapa”, no cabeçalho do arquivo há referências para o arquivo *CSS*, o ícone, as bibliotecas do *Firebase* e os códigos *Map* e *MapaDAO*. No corpo do *HTML* só precisa ser informado o *id* para ter seu estilo modificado pelo arquivo *CSS*.


```

resources > views > index.ejs > ...
1  <!DOCTYPE html>
2  <html lang="pt-br" id="corpoMapa">
3
4  <head>
5      <meta charset="utf-8" />
6
7      <title>Mapa</title>
8      <link rel="stylesheet" href="../css/Mapa.css">
9      <link href="../images/esp8266.png" rel="icon" type="image/x-icon">
10     <script src="https://www.gstatic.com/firebasejs/7.15.1/firebase-app.js"></script>
11     <script src="https://www.gstatic.com/firebasejs/7.15.1/firebase-analytics.js"></script>
12     <script src="https://www.gstatic.com/firebasejs/7.15.1/firebase-database.js"></script>
13     <script src="../js/MapaDAO.js"></script>
14     <script src="../js/Map.js"></script>
15     <style>
16
17     </style>
18 </head>
19
20 <body id="corpoMapa">
21     <div id="map"> </div>
22 </body>
23
24 </html>

```

Figura 18 - Arquivo Index no Código JavaScript

Para exibir a página somente é necessário carregar o código do arquivo inicial e acessar a porta informada no navegador pelo *localhost*. Como a porta informada foi a 3000, basta digitar <http://localhost:3000> no navegador que a página será exibida.

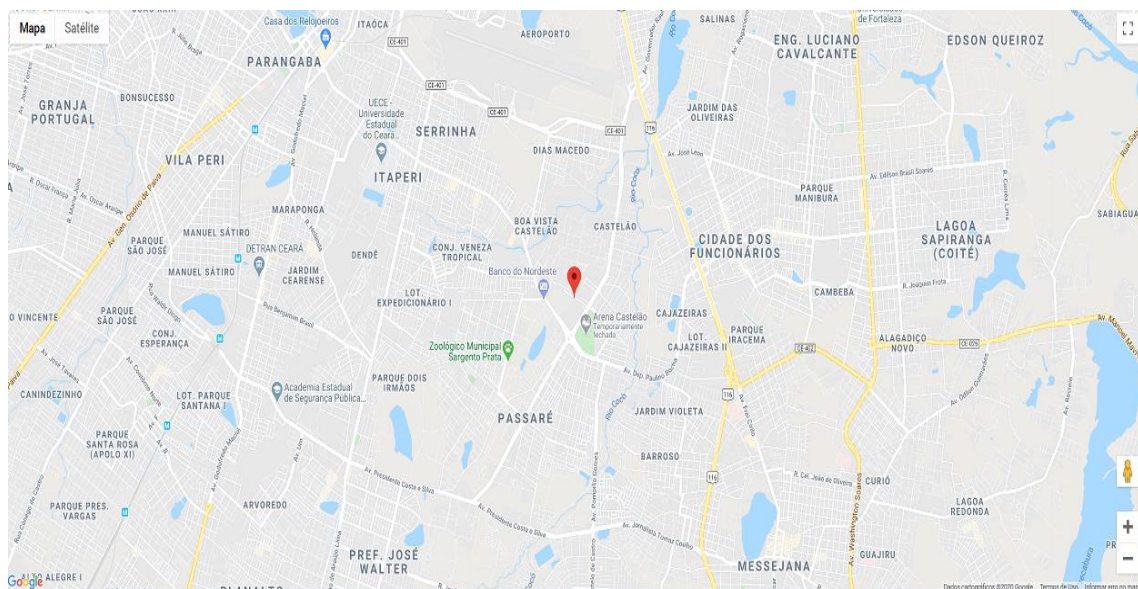
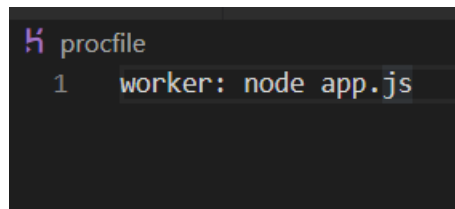


Figura 19 - Página mostrando o mapa

5. IMPLANTAR NO HEROKU

Para exibir e poder acessar a página de qualquer dispositivo e estando conectado a qualquer rede de forma gratuita, será utilizado o Heroku como explicado anteriormente.

Foi criado um projeto com o nome *rastreador-economico* e para armazenar o projeto na nuvem do Heroku é necessário enviar os arquivos para os servidores do mesmo. Mas antes é preciso adicionar um arquivo “procfile” na pasta com os códigos JavaScript para o heroku saber qual é o arquivo inicial. Dentro do procfile somente é necessário informar o tipo de arquivo e o nome do mesmo.



```
procfile
1 worker: node app.js
```

Figura 20 - Arquivo Procfile no Código JavaScript

Também é preciso definir a instância *start*, para novamente o Heroku entender qual é o arquivo inicial da aplicação. Assim ficará no final o arquivo *package.json*.



```
{ } package.json > ...
1 {
2   "name": "projeto-rastreador",
3   "version": "1.0.0",
4   "description": "",
5   "main": "app.js",
6   "dependencies": {
7     "ejs": "^3.0.1",
8     "express": "^4.17.1",
9     "firebase": "^7.15.1"
10  },
11  "devDependencies": {
12    "jquery": "3.3.1",
13    "popper.js": "1.14.3"
14  },
15  "scripts": {
16    "start": "node app.js",
17    "test": "echo \\\"Error: no test specified\\\" && exit 1"
18  },
19  "author": "",
20  "license": "ISC"
21 }
```

Figura 21 - Arquivo Package.json

Finalizada a configuração inicial do Heroku é necessária subir a aplicação utilizando os comandos informados no site do mesmo.

Install the Heroku CLI

Download and install the [Heroku CLI](#).

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

```
$ heroku login
```

Clone the repository

Use Git to clone rastreador-economico's source code to your local machine.

```
$ heroku git:clone -a rastreador-economico  
$ cd rastreador-economico
```

Deploy your changes

Make some changes to the code you just cloned and deploy them to Heroku using Git.

```
$ git add .  
$ git commit -am "make it better"  
$ git push heroku master
```

Figura 22 – Subindo aplicação no Heroku

Para verificar se o sistema está online basta acessar o nome do projeto com o final “.herokuapp.com”, neste caso como o utilizado foi *rastreador-economico*, a aplicação está online no endereço *rastreador-economico.herokuapp.com*.

6. CONCLUSÃO

Foi criado da forma mais econômica possível um rastreador utilizando somente um microcontrolador que recebe através das solicitações das redes WiFi a localização com as coordenadas e as envia para um servidor em tempo real, o Firebase, as informações mencionadas, assim como a hora e a data da última atualização. Para a exibição em uma página web é utilizado um servidor em nuvem gratuito, o Heroku.

O primeiro passo foi o carregamento da placa através do código arduino que fica atualizando em torno de um minuto a localização do dispositivo enviando as informações para o Firebase.

Após isso, foi necessário carregar essas informações para exibir em uma página da internet, assim, utilizou-se de JavaScript para criar funções para receber as informações do servidor e salvá-las para poder exibir com sucesso.

Por último foi feito o carregamento do projeto em JavaScript no Heroku para a visualização em outros dispositivos e redes ocorrer com sucesso.

7. REFERÊNCIAS

- OLIVEIRA, Euler. Conhecendo a Wemos D1. BLOG MASTERWALKER SHOP. Disponível em: <https://blogmasterwalkershop.com.br/embarcados/wemos/conhecendo-a-wemos-d1/>. Acesso em: 14/07/2020;
- MARTÍN, Germán. WiFi Location. GitHub. Disponível em: <https://github.com/gmag11/WifiLocation>. Acesso em: 15/07/2020;
- MDN Web Docs - Mozilla. Introdução Express/Node. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introdução. Acesso em: 20/07/2020;
- PINTO, Pedro. Tutorial Node.js: Como usar o engine EJS?. Medium, 07 de nov. de 2019. Disponível em: <https://medium.com/@pedrumpinto/tutorial-node-js-como-usar-o-engine-ejs-12bcc688ebab>. Acesso em: 20 de jul. de 2020;
- JAVED, Adeel. **Criando Projetos com Arduino para a Internet das Coisas**. São Paulo: Novatec: 2017.