Carro con geolocalización empleando un microcontrolador NodeMCU y un Módulo Neo 6M con comunicación a base de datos Firebase

Para el desarrollo de un prototipo de un carro que emplee geolocalización se hace necesario adquirir los siguientes componentes que harán parte fundamental del dispositivo:

1. Microcontrolador NodeMCU: Dispositivo que funciona por medio de conexión WiFi, cuya programación se realizará empleando el editor de código (IDE) de Arduino, empleando el lenguaje de programación del mismo: C++.
2. Módulo GPS Neo 6M: Módulo GPS de alta precisión que se conecta y comunica por medio de protocolo serial (RX y TX).
3. Puentes de conexión: Necesarios para la interconectividad de los dispositivos (microcontrolador y módulo GPS).

De igual forma, se hacen necesarios algunos componentes de software que permitirán almacenar y visualizar la información recopilada por el microcontrolador y el módulo GPS, los cuales son:

1. Base de datos Firebase: Se eligió éste modelo de base de datos, proporcionado por la multinacional Google a través de su servicio para desarrolladores Google Cloud Platform, ya que permite almacenar y leer valores en tiempo real entre diferentes protocolos de programación, en particular, entre C++ para almacenar desde el microcontrolador, y JavaScript / HTML para leer y mostrar la información.
   La base de datos requiere que exista una tarjeta de crédito vinculada a la cuenta de Google con la que se desee trabajar, esto debido a que, aunque se ofrece un espacio y ancho de banda decente de manera gratuita mensualmente, en caso de excederse puedan cobrarse los sobrecostos por el uso del aplicativo (el cual consume muy pocos recursos, ya que en un funcionamiento continúo de envío de datos y una lectura de un dispositivo puede consumir cerca del 1% del permitido en el plan gratuito)
2. Servidor web: Se hace necesario para poder mostrar la información leída por el dispositivo. En caso de contar previamente con un dominio, puede usarse para alojar el aplicativo el servicio gratuito de GitHub Pages, ya que consiste de un archivo HTML sin programación de back-end o del lado del servidor.

Para comenzar el desarrollo del prototipo se hizo necesario soldar las puntas del módulo GPS a los cables o puentes de conexión, y éstos a su vez asegurarlos a los pines definidos en la programación del microcontrolador, esto con el fin de garantizar cualquier pérdida de información debido a malas conexiones entre los dispositivos. Para darle una capa adicional de seguridad al dispositivo, se hizo necesario agregarle un pegamento aislante, específicamente silicona líquida, de tal forma que en caso de que fallaran las soldaduras de los cables, éste afianzara los puentes a los dispositivos, y adicionalmente brindara un aislante en caso de que un aumento de temperatura creara una conducción no deseada entre las soldaduras.

Luego se procedió a desarrollar un código en C++ para el microcontrolador, el cual se describe a continuación, asignando asteriscos a las conexiones con la base de datos, por motivos de seguridad.

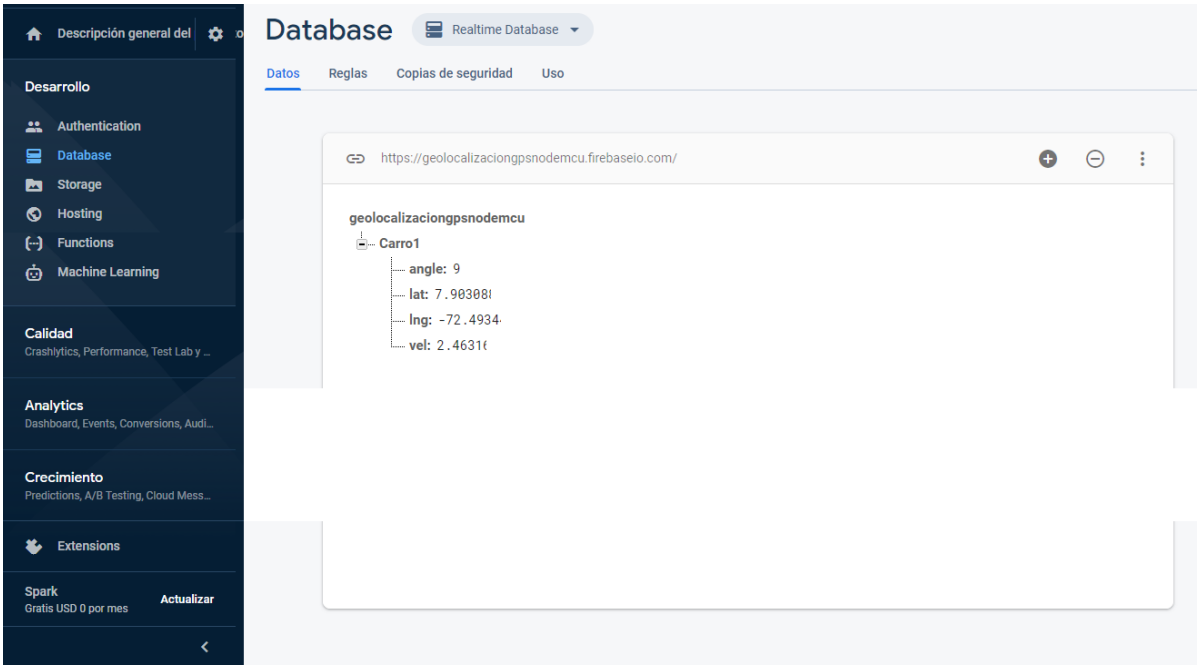
```
#include <ESP8266WiFi.h>
#include <FirebaseArduino.h>
#include <SoftwareSerial.h>
#include <TinyGPS.h>
#define FIREBASE_HOST " **********.firebaseio.com"
#define FIREBASE_AUTH " ****************************************** "
#define WIFI_SSID "Carro Movil"
#define WIFI_PASSWORD "clavefacil"
TinyGPS gps;
SoftwareSerial gps_serial(4, 5); // D1 y D2 || D1 a RX, D2 a TX (NodeMCU)
void setup() {
  Serial.begin(115200);
  gps_serial.begin(9600);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
}
void loop() {
   bool newData = false;
   unsigned long chars;
   unsigned short sentences, failed;
   for (unsigned long start = millis(); millis() - start < 1000;){
    while (gps_serial.available()) {
      int c = gps_serial.read();
      if (gps.encode(c)){
        newData = true;
      }
     }
  }
  if (newData) {
    float flat, flon, gpscourse;
    float fkmph = gps.f_speed_kmph();
    unsigned long age;
    gpscourse = gps.f_course();
    gps.f_get_position(&flat, &flon, &age);
    Firebase.setFloat("Carro1/lat", flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat);
    Firebase.setFloat("Carro1/lng", flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon);
    Firebase.setFloat("Carro1/vel", fkmph == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : fkmph);
    Firebase.setFloat("Carro1/angle", gpscourse == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 :
gpscourse);
   }
}
```

En caso de no contar con experiencia en programación con C++ o con microcontroladores en general, el código hace lo siguiente de manera consecutiva:

1. Incluir las librerías necesarias para su funcionamiento (microcontrolador, conexión con Firebase, comunicación serial y módulo GPS).
   Nota: La librería de comunicación de Firebase deberá actualizarse cada año por motivos de seguridad y por tanto actualizar el código del microcontrolador en el mismo periodo de tiempo. De no hacerlo, el aplicativo dejará de funcionar parcial o totalmente.

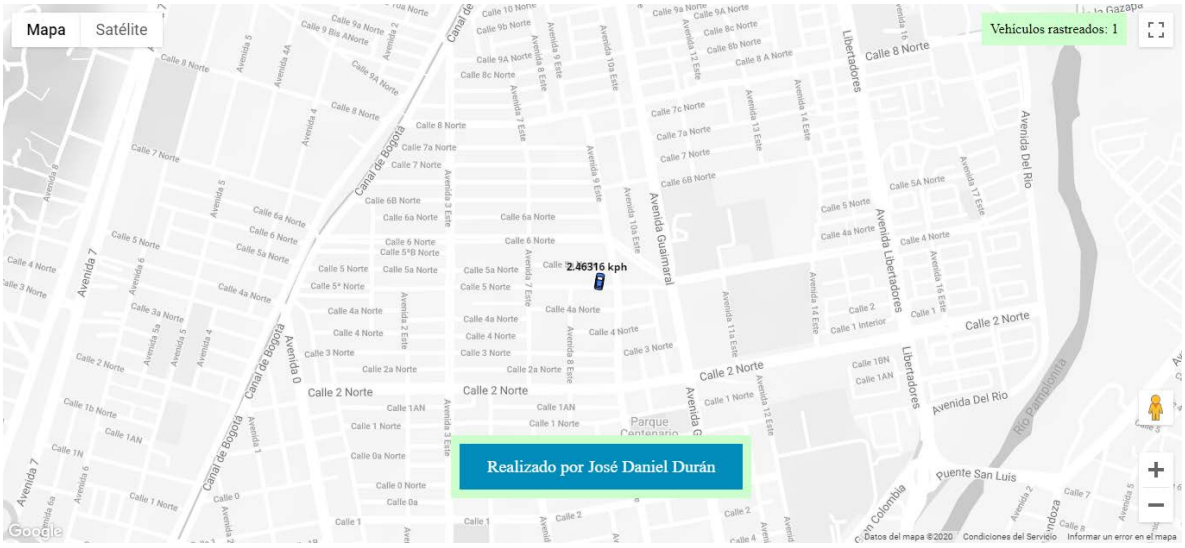
2. Definir las variables de conexión:
    a. FIREBASE_HOST: URL del servidor provista por Firebase al momento de configurar la base de datos.
    b. FIREBASE_AUTH: Clave única de autenticación (privada) con la cual se comunicará el microcontrolador con la base de datos Firebase, se genera al momento de configurar la base de datos.
    c. WIFFI_SSID: Nombre de la red WiFi a la cual se conectará el dispositivo (lo cual reintentará constantemente hasta encontrar una red WiFi con este nombre)
    d. WIFI_PASSWORD: Clave de la red WiFi a la cual se conectará el dispositivo (lo cual reintentará constantemente hasta encontrar una red WiFi con el nombre descrito anteriormente y que coincida con esta clave)
3. Define los pines físicos del microcontrolador con los cuales se encuentra conectado el dispositivo al módulo GPS. Por razones de diseño, la configuración de pines es la siguiente.
    a. D1 descrito físicamente equivale a un 4 internamente en el dispositivo. En este pin recibirá la información del módulo GPS.
    b. D2 descrito físicamente equivale a un 5 internamente en el dispositivo. En este pin enviará información al módulo GPS confirmando la recepción de los datos.
4. En el “Setup” se encuentra la configuración inicial del dispositivo, en la cual realiza las conexiones y validaciones iniciales.
5. En el “Loop” realiza una continua verificación del dispositivo GPS, esperando recibir datos del mismo. Al recibir datos de éste, envía una señal de confirmación en el código para que realice la toma de datos y el envío a la base de datos de Firebase.
    a. Carro1/lat en Firebase almacena la latitud del dispositivo.
    b. Carro1/lng en Firebase almacena la longitud del dispositivo.
    c. Carro1/vel en Firebase almacena la velocidad actual del dispositivo.
    d. Carro1/angle en Firebase almacena el posible ángulo actual del dispositivo (en qué dirección está mirando). Este dato no es del todo preciso ni exacto.

Los valores almacenados se muestran de la siguiente manera en la base de datos de Firebase:



Confirmando la recepción de datos en Firebase, se procedió a desarrollar una página web que permitiera visualizar la información almacenada.

Empleando HTMl, CSS y JavaScript, y técnicas de manejo de los mapas de Google (Google Maps) para la ubicación de marcadores y punteros en el mismo, se desarrolló el siguiente demo funcional del dispositivo:



https://carro.josedanield10.com/

El código y la explicación del mismo para la visualización de los vehículos y su “movimiento en el mapa” se encuentran fuera del alcance y propósito de este proyecto.

Datos relevantes a modo de operación:

- La red WiFi debe nombrarse como “**Carro Movil**” (sin tildes, con las mayúsculas propuestas)
- La red WiFi debe tener la siguiente clave: “**clavefacil**” (sin espacios, sin tildes)

En caso de desear desarrollar más dispositivos, se deben cambiar las credenciales de Firebase y actualizar el código para que reflejen los nuevos vehículos (Carro1, Carro2, Carro3, Carro**N**)