ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"DISEÑO DE UN SISTEMA DE ADQUISICIÓN, TRANSMISIÓN Y ANÁLISIS DE DATOS VEHICULARES, EN BASE A SENSORES, PARA LA GESTIÓN Y SEGURIDAD DEL TRÁFICO"

INFORME DE MATERIA INTEGRADORA

Previo a la obtención del Título de:

INGENIERO EN TELECOMUNICACIONES

PAUL ANDRÉS AGUILAR MACANCHÍ

CRISTHIAN MARCELO PACALLA GUAMÁN

GUAYAQUIL – ECUADOR

AÑO: 2017

AGRADECIMIENTOS

Al finalizar una etapa en el largo proceso educativo, es justo agradecer a las personas que confiaron en mí, que brindando su apoyo de manera incondicional a través de palabras y acciones hicieron estos años más placenteros, logrando de esta forma cumplir uno de mis objetivos propuestos y a la vez iniciar una nueva etapa en mi vida.

A mis amigos con los que compartí muchas risas y angustias desde el primer momento en que ingresé a la universidad, al MSc. Cesar Yépez por sus indicaciones y consejos en el proceso de graduación, a mis familiares que supieron animarme y aconsejarme y principalmente a mis padres, quienes fueron incondicionales en todo momento y que de manera divertida esperaron este momento más que mí.

Paul Andrés Aguilar Macanchí

Mis más sinceros agradecimientos primero a Dios por sus Bendiciones que derraman a mi familia, mis Padres por su esfuerzo lograron que termine esta etapa de mi vida, mis amigos desde el Preuniversitario que hicieron que mi vida estudiantil sea una aventura y al MSc. Cesar Yépez por su contribución para la realización de esta tesis.

Cristhian Marcelo Pacalla Guamán

DEDICATORIA

Dedico este trabajo a mis amigos y familiares que incondicionalmente mostraron su apoyo de manera incondicional sin pedir algo a cambio.

A mis abuelitas, las señoras Rosa Elida y Úrsula que en toda oportunidad supieron brindarme su apoyo y aconsejarme de la mejor manera posible.

A mis tíos Dora, María y Fernando que de distintas maneras supieron compartir sus experiencias y su apoyo para que no me detenga y siguiera adelante.

Y a mis padres, Lucía y Wilson que incondicionalmente estuvieron presentes en todo momento, brindando su apoyo y aconsejándome para que supiera aprender de mis errores y de esta manera poder hacerle frente a la vida.

Paul Andrés Aguilar Macanchí

El presente proyecto lo dedico primeramente a Dios por guiar mis pasos hasta la culminación de mi carrera, mis Padres Mercedes y Luis por su apoyo incondicional, motivación, palabras de aliento y sobre todo el cariño que me motivaron a que este sueño se pueda realizar, mis hermanos por su aporte durante mis estudios universitarios y a mi familia en general que con sus obras lograron que pudiera cumplir una de mis metas, sabiendo que con esfuerzo, dedicación y sobre todo amor a lo que haces pueda concluir esta etapa de mi vida.

Cristhian Marcelo Pacalla Guamán

TRIBUNAL DE EVALUACIÓN

MSc. César Yépez Flores

PROFESOR EVALUADOR

PhD. Francisco Novillo Parales

PROFESOR EVALUADOR

DECLARACIÓN EXPRESA

"La responsabilidad y la autoría del contenido de este Trabajo de Titulación, nos corresponde exclusivamente; y damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Paul Andrés Aguilar Macanchí

Cristhian Marcelo Pacalla Guamán

RESUMEN

Este proyecto busca encontrar una alternativa más efectiva al momento de detectar problemas como robos o accidentes en vehículos de transporte público, de esta manera prevenir consecuencias como muertes y/o pérdidas materiales proporcionando ayuda inmediata a través de una central de auxilio.

Este proyecto consta de dos partes: un controlador y un servidor de datos.

El controlador está conformado por: un lector de tarjeta, que servirá para la identificación del chofer del vehículo de transporte público; un módulo GPS (en español, Sistema de Posicionamiento Global), que brindará la posición y velocidad del vehículo en tiempo real; dos botoneras, utilizadas para las llamadas de auxilio en casos de robos o accidentes respectivamente; un módulo Arduino, que será el encargado de administrar y ordenar los datos; una pantalla LCD (en español, Pantalla de Cristal Líquido), la cual será el medio de interacción con el usuario; un módulo GPRS (en inglés, General Packet Radio Service), que será el encargado de enviar los datos al servidor.

El servidor de datos es el destinado a almacenar los datos provenientes del controlador. La visualización de los datos será por medio de una aplicación web y para su ingreso se deberá tener rango de administrador. En esta aplicación el administrador tendrá detalles del vehículo de transporte público, cómo su ubicación, el estado de la señal de auxilio y quien lo conduce. De esta manera, el administrador será libre de tomar la decisión de enviar la ayuda correspondiente al momento de una llamada de auxilio.

ÍNDICE GENERAL

CAPÍTULO 1

1. INTRODUCCIÓN.

1.1 Antecedentes.

Según datos proporcionados por la ANT (Agencia Nacional de Tránsito) en su página web, en el primer trimestre del presente año (2017) se han registrado 7.123 accidentes de tránsito sin consecuencias mortales en todo el país, de los cuales menos del 5% de accidentes han sido causados por factores externos, lo que nos deja con más del 95% como principal factor al error humano, teniendo como factor común el abandono del lugar por parte del conductor y dejando a la deriva las consecuencias de sus acciones, como daños materiales o pérdidas humanas [1].

1.2 Justificación.

Las infracciones de tránsito en las calles y carreteras ecuatorianas son a diario, lo preocupante de esto es que una gran parte de estas infracciones son cometidas en áreas urbanas por choferes del transporte público que no representan las normas de tránsito y a esto sumado el creciente índice delictivo crean molestias en los pasajeros, peatones y en el tráfico vehicular. Mientras que estos problemas cometidos en vías inter-cantonales o inter-provinciales quedan sin consecuencias debido a la lenta respuesta por parte de las autoridades competentes.

Estas infracciones no solo son un problema de tránsito sino un problema para la sociedad, el cual es necesario erradicar con fuertes sanciones a los respectivos causantes. En esta tesis se propondrá una solución a este problema, buscando crear una pauta para mejorar la seguridad de los pasajeros y mejorar la calidad de tránsito vehicular dentro y fuera de la ciudad.

1.3 Objetivos.

1.3.1 Objetivo General.

Desarrollar un sistema que permita mejorar el tiempo de reacción ante problemas puntuales como accidentes y/o robos en vehículos de transporte público a través de sensores y una base de datos, evidenciando a los posibles culpables y de esta manera aumentar la confianza de los usuarios en el transporte público.

1.3.2 Objetivos Específicos.

- Establecer un servidor en la nube que permita almacenar los datos adquiridos en los vehículos de transporte público en tiempo real para tener evidencia de las infracciones cometidas en caso de ocurrir accidentes de tránsito, robos o exceso del límite de velocidad.
- Implementar un modo de visualización de los datos almacenados en los servidores para garantizar una respuesta inmediata ante potenciales eventos que involucren daños o perjuicios para los pasajeros y peatones.
- Mejorar la seguridad de los pasajeros y el tránsito vehicular con un control de datos permanente de los vehículos de transporte público, con botones de acción inmediata y tiempo de respuesta.
- Implementar a futuro la recepción y visualización de datos en vehículos privados para mejorar la educación vial en las vías públicas.

1.4 Metodología.

La implementación de un servidor de base de datos en la nube permite almacenar y visualizar los datos incorporados al sistema el cual facilitará la ayuda a los agentes de tránsito y a los operarios de la plataforma a identificar a los causantes ya sea el caso de accidentes o robo.

A través de un módulo GPS (en español, Sistema de Posicionamiento Global) se procederá a obtener la velocidad y posición del vehículo en tiempo real. Se

dispondrá de un servicio de identificación para el chofer y el vehículo mostrando el nombre y el número de usuario del chofer, el número del vehículo y la cooperativa de transporte a la que pertenece. Tendrá incorporado dos botones de acciones externas a ejecutar por parte del chofer al momento de tener escenarios que involucren los siguientes riesgos humanos: accidente o robo, estas opciones estarán conectadas directamente al controlador por un medio cableado y estarán ubicados cerca del asiento del chofer.

Una central mostrará a través de una interfaz adecuada para un operario capacitado los datos adquiridos en tiempo real transmitidos de manera inalámbrica por medio de la red celular usando un módulo GPRS (en inglés, General Packet Radio Service) desde un controlador ubicado dentro del vehículo de transporte público y será el encargado de administrar los datos de interés del vehículo.

Con los resultados reales obtenidos en un periodo de tiempo, proponer esta iniciativa para la futura implementación en vehículos livianos y pesados, públicos y privados para un mejor control de tránsito vehicular.

1.5 Resultados Esperados.

Al finalizar este proyecto de investigación se espera contar con una central de control que reciba datos en tiempo real de las señales de auxilio enviadas desde un controlador ubicado dentro de un vehículo de transporte público y de esta manera poder enviar la ayuda necesaria para evitar pérdidas humanas y/o materiales. Además de mejorar el tránsito y la seguridad vial, generando más confianza en el transporte público y buscar una futura implementación a nivel nacional.

CAPÍTULO 2

2. SERVIDOR DE DATOS.

2.1 Lenguaje SQL.

SQL (en español, Lenguaje de Consulta Estructurada), es un lenguaje de programación estándar que permite la manipulación de datos de manera dinámica a través de tablas. Aunque SQL es a la vez una norma ISO (en español, Organización Internacional de Normalización), muchos productos de bases de datos soportan SQL con extensiones propietarias al lenguaje estándar. Las consultas toman la forma de un lenguaje de comandos que permiten seleccionar, insertar, modificar y eliminarla ubicación de datos dentro de una tabla [2].

SQL posee las siguientes características [3]:

- Lenguaje de definición de datos, proporciona comandos para la definición de esquemas de relación, borrado de relaciones.
- Lenguaje interactivo de manipulación de datos, esto incluye lenguajes de consultas basado tanto en comandos básicos de llamado a una librería especifica como en calculo relacional de tuplas.
- Integridad, permite la restricción de integridad que debe cumplir los datos almacenados en la base de datos.
- Definición de visitas, esto nos permite dar permisos a ciertos usuarios.
- Control de transacciones, comandos que especifican el comienzo y final de dicho proceso.
- Incorporación de instrucciones para lenguajes ya definidos por SQL, como es C++. C, JAVA, HTML, etc.

2.2 Creación de una base de datos en SQL.

Para utilizar el servidor de datos es necesario tener instalado el programa phpMyAdmin en nuestro servidor local, la instalación es a través de su página web: https://www.phpmyadmin.net

Para el acceso a la base de datos se ingresa a través de un usuario y contraseña configurados previamente en la instalación. Ver Figura 2.1.

Figura 2.1: Ingreso a phpMyAdmin.

Dentro de la aplicación el usuario es libre de agregar las tablas y los campos que crea necesarios. Ver Figura 2.2.

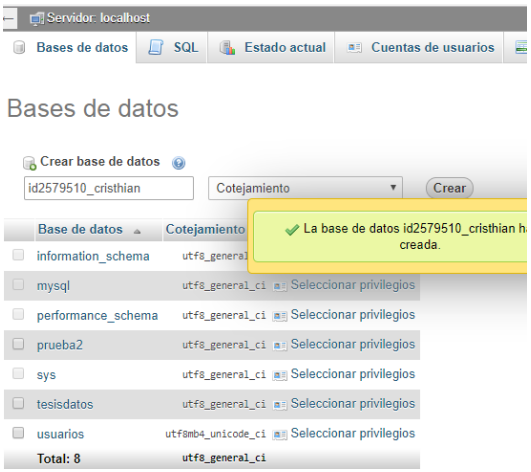

Figura 2.2: Creación de bases de datos en phpMyAdmin.

En la Figura 2.3 se observa la base de datos con sus respectivas tablas. La tabla consulta contiene toda la información del usuario desde sus nombres hasta su dirección domiciliaria. Estos datos que pueden ser utilizados por agentes de seguridad si el caso lo amerita.

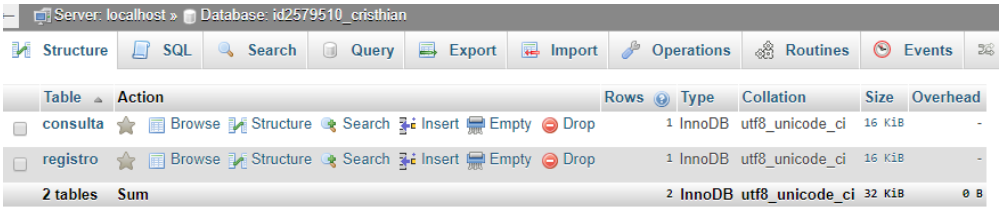

Figura 2.3: Registro de tablas creadas en phpMyAdmin.

Para guardar información de los usuarios se puede utilizar muchas plataformas de base de datos como ya lo mencionamos anteriormente, pero su fácil conexión entre su programa con la de una página web nos llevó a la utilización de la misma, puesto que otras plataformas se dificulta la programación para su correcta manipulación de datos.

2.3 Creación de una página Web.

Realizar la comunicación entre una plataforma SQL con un servidor Web es muy accesible si se utiliza el lenguaje de programación PHP (en inglés, Hypertext Preprocessor). El lenguaje PHP es ideal para el diseño de sitios web de contenido dinámico permitiendo la creación de aplicaciones con interfaces amigables para el usuario. PHP permite conectar la base de datos con un dominio público y/o privado con un servidor de datos que en nuestro caso es MySql para su posterior visualización en un servidor web.

Características de PHP:

- Creaciones de páginas web con interfaz dinámica además de obtener acceso a datos guardados desde una base de datos.
- Al momento de la creación de un servidor web su código principal es invisible para un cliente.

- Tiene licencia gratuita, esto permite la creación de páginas web y de fácil acceso para todos los usuarios si desean crear un servidor.
- Se puede utilizar programación orientada a objetos para un diseño más amigable.

PHP a diferencia de otros lenguajes la forma de ejecutar permite el acceso a datos guardados en plataformas, base de datos, que son visualizados en páginas HTML.

2.4 Enlazar una base de datos y una página web.

En la Figura 2.4 se aprecia un ejemplo de cómo enlazar una base de datos a PHP para su conexión y posterior manejo de información.

/public_html/conexion.php

```
<?php
$conect=mysqli_connect("localhost","id2579510_cpacalla","cmPG101991") or die ("NO SE ENCONTRO LA BASE DE DATOS");
mysqli_select_db($conect,"id2579510_cristhian") or die ("NO SE ENCONTRO LA BASE DE DATOS");
return $conect;
?>
```

Figura 2.4: Lenguaje de programación PHP.

PHP permite la búsqueda dinámica en el contenido de tablas permitiendo actualizar la información de la aplicación web según sea conveniente. En la Figura 2.5 se muestra un ejemplo de la búsqueda de contenido en una tabla dentro de una base de datos a través de PHP.

```
<?php
}
?>
<td align="center"><?php echo $n; ?>
<td align="center"><?php echo $reg[1]; ?>
<td><?php echo $reg[2];?>
<td align="center"><?php echo $reg[3]; ?>
<td align="center"><?php echo $reg[4]; ?>
<td align="center"><?php echo $reg[5]; ?>
<td align="center"><?php echo $reg[6]; ?>
<td align="center"><?php echo $reg[7]; ?>
<td align="center"><?php echo $reg[8]; ?>
<td align="center"><?php echo $reg[9]; ?>
<td align="center"><a class="button mediano naranja" onclick="window.location='mapa.php'"><span>MAPA</span></a>
<td align="center"><?php echo $reg[11]; ?>
<td align="center"><?php echo $reg[12]; ?>
<td align="center"><?php echo $reg[13]; ?>
<?php
```

Figura 2.5: Búsqueda dinámica de datos a través de PHP a una base de datos.

Al encontrar los datos en los campos correspondientes, PHP actualiza los datos de la aplicación web a la cual sea enlazado. Ver Figura 2.6.

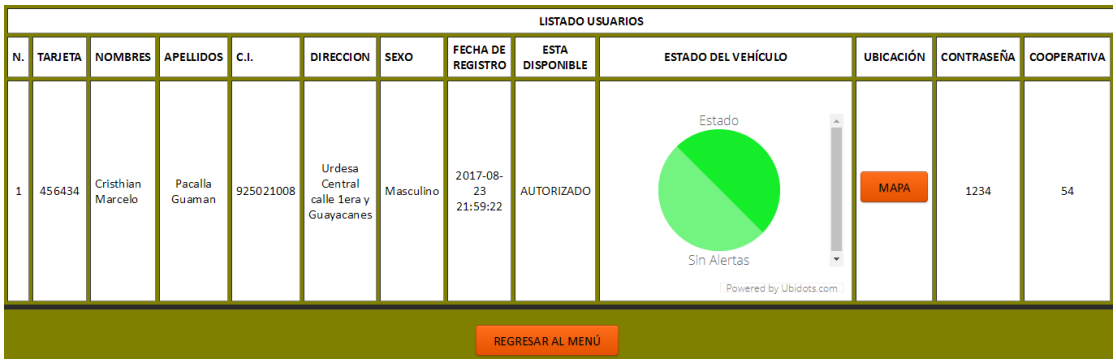

Figura 2.6: Visualización de datos almacenados en un servidor a través de una página web.

2.5 UBIDOTS.

Cómo se muestra en la Figura 2.7, Ubidots es un servicio en la nube que permite almacenar e interpretar información obtenida en tiempo real, haciendo posible la creación de aplicaciones para el IoT (en español, Internet de las Cosas) de una manera rápida y sencilla [4].

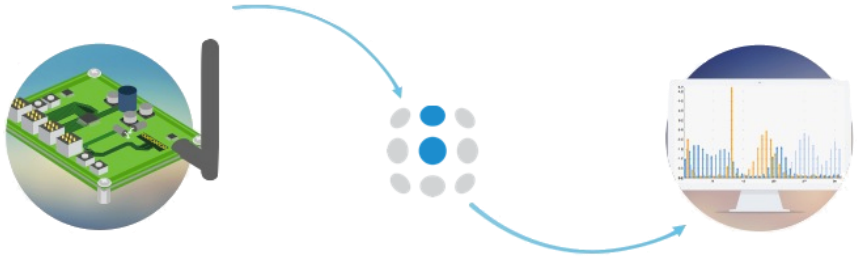

Figura 2.7: Funcionamiento del IoT en Ubidots.

Ubidots dispone de 4 términos indispensables para el manejo de datos en su nube: Data source, Variable, Value y Event.

- Data Source: Es un conjunto de información que hace referencia a un dispositivo, en la cual cada data source puede tener una o más variables.
- Variable: Son un conjunto de datos que pueden cambiar con el tiempo.
- Value: Es el valor presente de una variable en un determinado momento.
- Event: Un evento es una acción a tomar en algún determinado momento.

Ubidots permite interactuar con cada uno de estos elementos a través de su API (Application Programming Interface) de una manera programática, es decir que estos elementos pueden ser creados, modificados o eliminados.

2.5.1 API de Ubidots.

El API de Ubidots implementa los cuatro principales métodos de HTTP:

- GET: Dedicado a la lectura de información.
- POST: Utilizado por la creación de información.
- PUT: Utilizado para la edición de información.
- DELETE: Utilizado en la eliminación de información.

Para el uso de la API de Ubidots se debe especificar el formato de los datos a enviar. El formato más común es el JSON (JavaScript Object Notation) debido a su simplicidad. Ver Figura 2.8.

```
JSON

{
  "value": 1000, "context": { "estado":
  "encendido" }

}
```

Figura 2.8: Formato JSON para envío de paquetes de datos.

2.5.2 Interacción con el API de Ubidots.

Para crear una partición en la nube de Ubidots tenemos que crear un usuario en su servidor, esto se hace directamente desde su página principal. Ver Figura 2.9.

Figura 2.9: Pantalla principal de la página web de Ubidots.

2.5.3 Creación de un Data Source.

En el panel principal de Ubidots hacer clic en la opción Dispositivos/Agregar Dispositivo y procedemos a agregar un nombre. Se puede agregar tantas Data Source como sea necesario. En este caso se creará una que contendrá todas las variables de interés. Ver Figura 2.10.

Figura 2.10: Creación de una Data Source.

2.5.4 Creación de una Variable.

En el panel de Dispositivo hacer clic en nueva variable y se procede a colocar el nombre de la misma. Ver Figura 2.11.

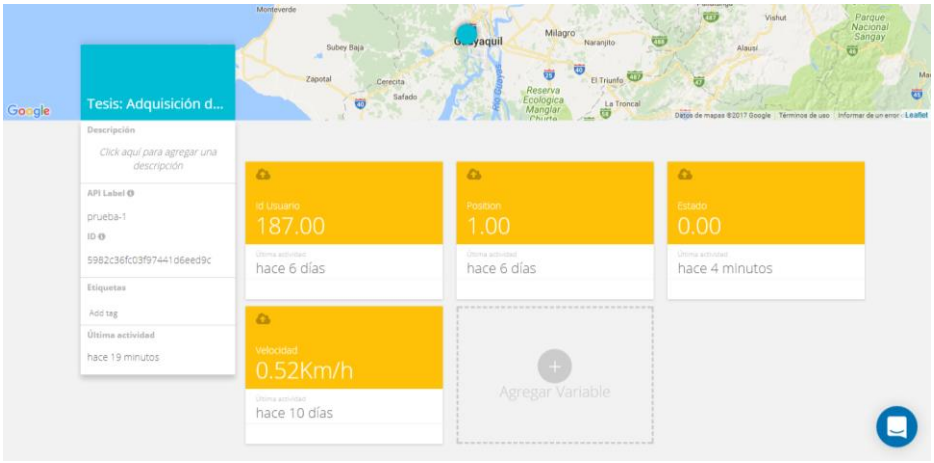

Figura 2.11: Variables creadas dentro de nuestro Data Source.

2.5.5 Enviando datos a una Variable en Ubidots.

Para el envío de datos a una variable dentro de Ubidots necesitamos saber dos cosas: Como identificar a la variable y cómo acceder a la nube.

Cómo se muestra en la Figura 2.12, Ubidots proporciona un ID en cada variable y un token por cada cuenta.

El ID es la identificación de cada variable dentro de la nube, se encuentra dentro de cada una y se crea uno por cada variable dentro de cada Data Source.

El token es la llave de acceso a la nube, se encuentra en la casilla de credenciales y se puede generar más de uno según sea conveniente.

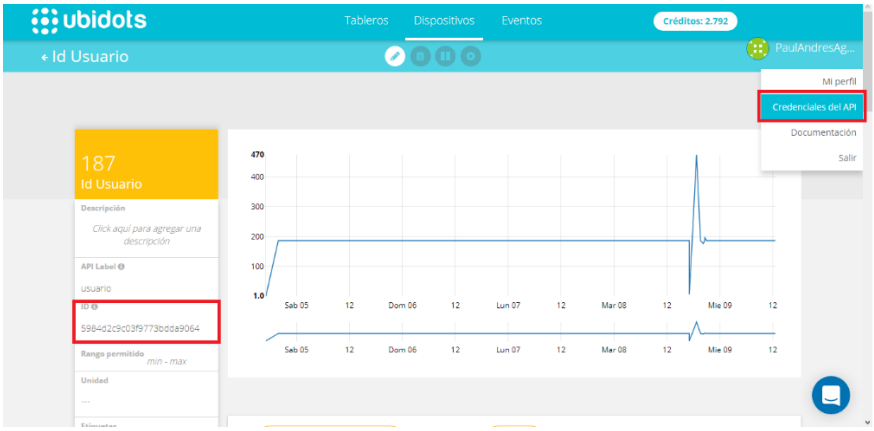

Figura 2.12: Identificación de ID y token en Ubidots.

El envío de datos debe tener estas dos identificaciones para que Ubidots los pueda almacenar correctamente. El formato del empaquetamiento de datos por el método JSON para el envío de datos es el siguiente:

- POST /api/v1.6/variables/{ID}/values HTTP/1.1
- Host: things.ubidots.com
- Content-Type: application/json
- X-Auth-Token: {token}
- Content-Length: {Longitud total de la cadena value}
- {"value":1000}

En donde {ID} y {token} son los valores proporcionados por Ubidots sin llaves y Content-Length: es la longitud total de value.

2.5.6 Presentación de una variable en Ubidots.

Ubidots permite presentar las variables en su Dashboard en forma de widgets. Un widget es una aplicación cuya función es proveer información visual de algún dato en particular y que a la vez pueden ser insertados en otras páginas web a través de iframe's (en inglés, inline frame).

Para crear un widget hacer clic en la opción nuevo widget. Luego se presentará una variedad de widgets que pueden ser usados (elegir la presentación más adecuada al tipo de variable a mostrar). Finalmente enlazar la variable al widget. Ver Figura 2.13.

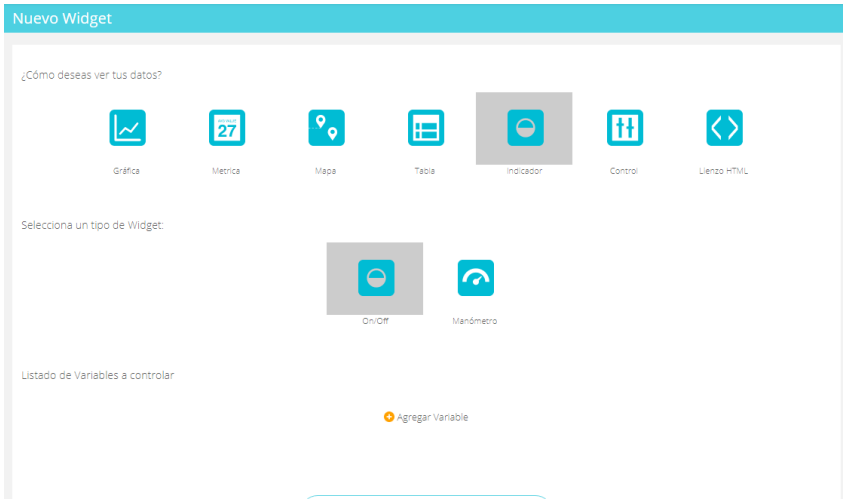

Figura 2.13: Creación de un widget del Dashboard de Ubidots.

Ubidots permite crear tantos widgets como sea necesario por cada variable. Se tiene total libertad en la posición y el tamaño de los widgets para dar un mayor entendimiento del panel. Ver Figura 2.14.

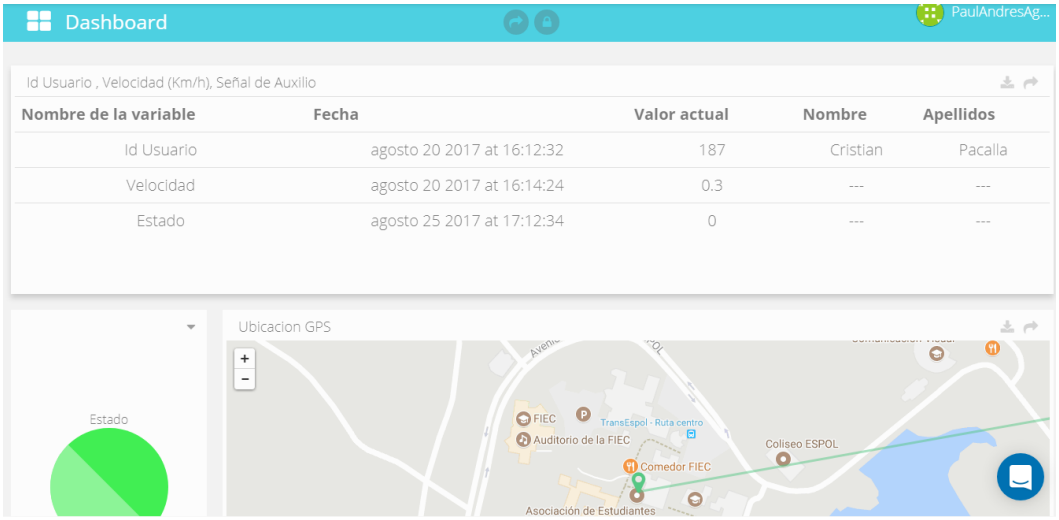

Figura 2.14: Presentación del Dashboard de Ubidots.

Para compartir cada widget en forma de iframe basta con hacer clic en el botón compartir ubicado en cada widget. De ser necesario, también se puede compartir el iframe del Dashboard completo dentro de otra página web. Ver figuras 2.15 y 2.16 respectivamente.

Figura 2.15: Iframe de la variable ubicación en Ubidots.

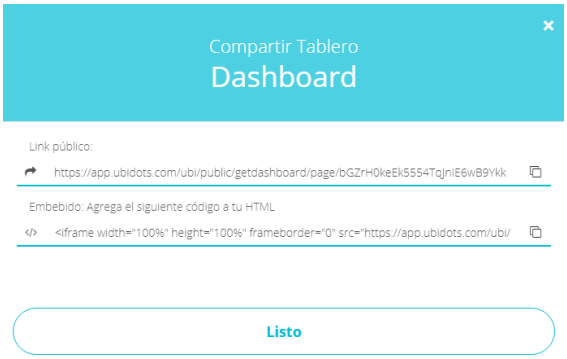

Figura 2.16: Iframe del Dashboard de Ubidots.

CAPÍTULO 3

3. ADQUISICIÓN Y CONTROL DE DATOS VEHICULAR.

Un controlador (Arduino) será el encargado de administrar los datos y señales obtenidos por el sensor de ubicación, el lector de tarjeta magnética y las botoneras. Los datos de velocidad y posición obtenidos a través de un módulo GPS serán relacionados con los datos del vehículo y el chofer obtenidos por parte del lector de tarjeta magnética. Las botoneras al ser presionadas enviarán sus señales correspondientes: “Accidente” o “Robo”. Estos datos serán presentados por una pantalla LCD que será el medio de interacción entre el controlador y el usuario. El módulo GPRS es el encargado de enviar los datos a través de la red celular a un servidor (nube), en donde la nube es la encargada del almacenamiento y la visualización de los mismos. Finalmente, estos datos podrán ser visualizados en una página web, un celular con sistema Android o en la misma nube. Ver Figura 3.1.

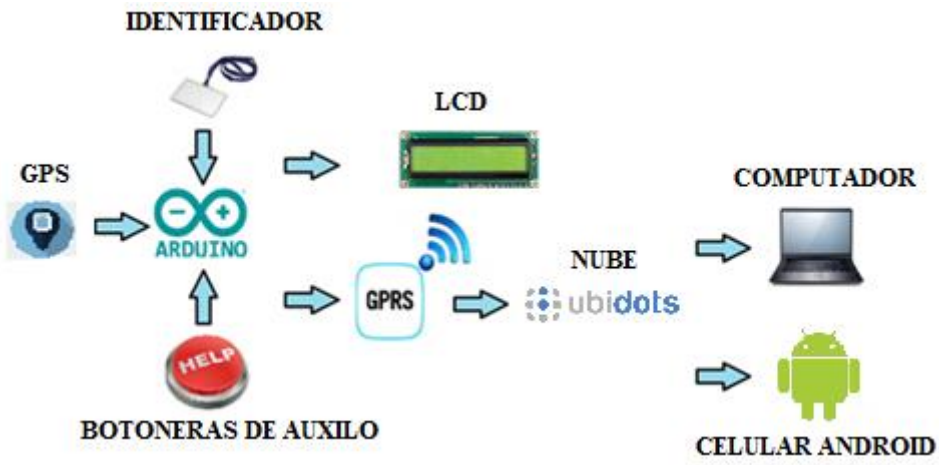

Figura 3.1: Representación gráfica de la solución al problema propuesto.

3.1 ARDUINO MEGA 2560.

Cómo se muestra en la Figura 3.2, el Arduino Mega es una tarjeta de desarrollo open-source construida con un microcontrolador modelo Atmega2560 que posee pines de entradas y salidas (E/S), analógicas y digitales. Esta tarjeta es programada en un entorno de desarrollo que implementa el lenguaje Processing/Wiring. Arduino puede utilizarse en el desarrollo de objetos interactivos autónomos o puede comunicarse a un PC a través del puerto serial (conversión con USB) utilizando lenguajes como Flash, Processing, MaxMSP, entre otras [5].

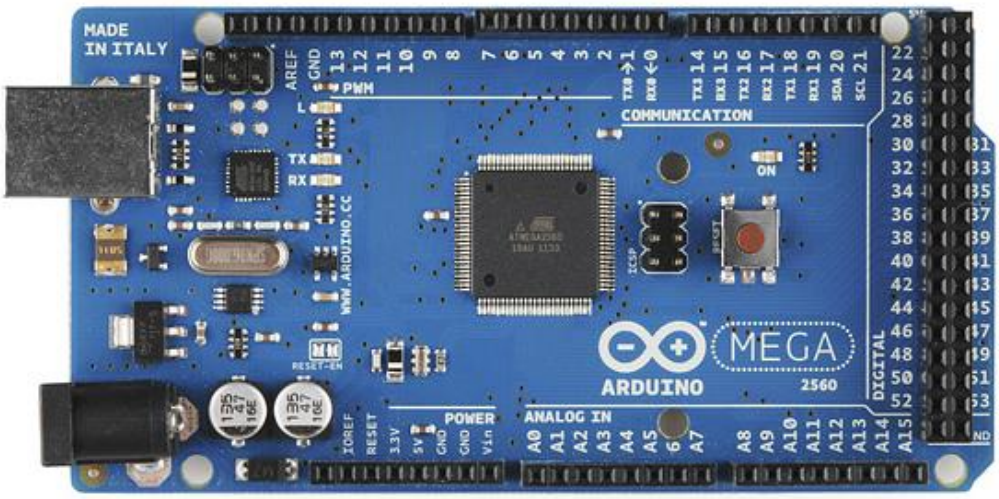

Figura 3.2: Módulo Arduino MEGA 2560.

El Arduino Mega posee sus propias especificaciones: cuenta con entradas y salidas analógicas/digital, EEPROM, clock interno, entre otras. Ver Tabla 1 [6].

ESPECIFICACIONES DE LA PLACA ARDUINO	
Voltajes de salida	**5 V y 3.3 V**
Voltaje de entrada (recomendado)	9 – 12 V
Pines In/Out digitales	54
Pines de entrada analógicas	16
Corriente de pines In/Out	40mA
Corriente de pin de 3.3 V	50mA
Memoria Flash	256 KB
SRAM	8 KB
EEPROM	4 KB
Velocidad reloj interno	16 MHz

Tabla 1: Especificaciones de la Placa Arduino Mega.

Arduino se programa a través de su Entorno de Desarrollo Integrado (IDE). Su lenguaje de programación es C++ y es posible instalar librerías para optimizar su uso con dispositivos externos como GPS, bluetooth, GPRS, entre otros.

La programación en el IDE consta de dos partes fundamentales el void setup() y el void loop(). El void setup() es la parte en donde se especifica las características de la placa como los pines de entrada/salida, frecuencia de trabajo, mientras el void loop() es la parte en donde se programa el comportamiento de la placa. Ver Figura 3.3.

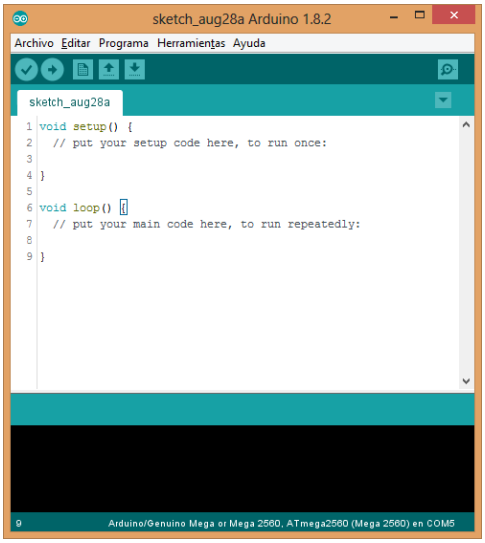

Figura 3.3: Interfaz del IDE de Arduino.

3.2 LCD 16x2 - EW162B0YMY.

La LCD 16×2 - EW162B0YMY es una pantalla de 2 filas con 16 columnas cada una. Funciona con un microcontrolador interno que es el encargado de convertir las señales recibidas en los caracteres a mostrar. Ver Figura 3.4.

La LCD posee sus propias especificaciones como voltaje, sincronización de datos entre otros. Ver Tabla 2 [7].

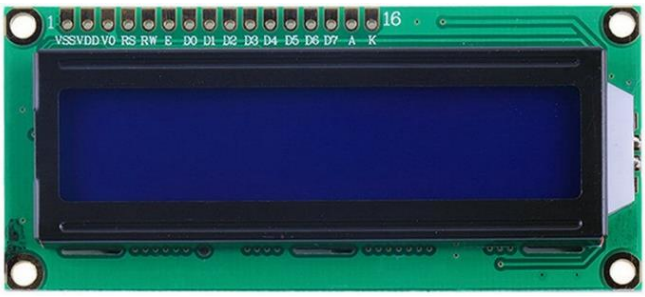

Figura 3.4: LCD 16x2 - EW162B0YMY [8].

PINES LCD 16x2	DESCRIPCIÓN
1	Tierra del módulo.
2	Alimentación del LCD 5 V.
3	Control de brillo de pantalla
4	RS – Selector entre comandos y datos
5	RW – Escritura y lectura de comandos y datos.
6	Sincronización de lectura de datos.
7 - 14	Pines de bus de datos de 8-bit
15	Alimentación luz de fondo (5V)
16	GND (Tierra) luz de fondo (0V)

Tabla 2: Descripción de los Pines del LCD 16x2 - EW162B0YMY.

3.3 Módulo GPS GY-GPS6MV2.

El módulo GPS en su modelo GY-GPS6MV2 viene con un módulo de serie U-Blox NEO 6M equipado en el PCB, una EEPROM con configuración de fábrica, un indicador LED y una antena cerámica. Ver Figura 3.5.

El Módulo GPS posee los pines o conectores Vcc, Rx, Tx y Gnd por el que es posible tener una conexión a un controlador mediante una interfaz serial. Ver Tabla 3 [9].

Figura 3.5: Módulo GPS GY-GPS6MV2. [10]

MÓDULO GPS GY-GPS6MV2.	DESCRIPCIÓN
GND	Tierra del módulo.
Vcc	Alimentación del módulo entre 3.3 V y 5 V.
Tx	Pin para la transmisión de datos. Va conectado al pin RX1 del módulo Arduino Mega.
Rx	Pin para la recepción de datos. Va conectado al pin Tx1 del módulo Arduino Mega.

Tabla 3: Descripción de los Pines del Módulo GPS GY-GPS6MV2.

3.4 Módulo RFID-MFRC522 RF.

El Módulo Lector RFID-RC522 RF (Radio Frecuency IDentificaton) funciona con un voltaje de 3.3V y se maneja por medio de una interfaz SPI (Serial Peripheral Interface) lo que lo hace compatible con la mayoría de microcontroladores. El RC522 utiliza un sistema avanzado de modulación y demodulación para todo tipo de dispositivos pasivos de 13.56Mhz. Puesto que posee un sistema de lectura/escritura lo que le permite modificar los datos de la tarjeta es necesario conocer las características de los bloques de memoria una tarjeta: La tarjeta que viene con el módulo RFID cuenta con 64 bloques de memoria (0-63) donde se hace lectura y/o escritura. Cada bloque de memoria tiene la capacidad de almacenar sobre todo hasta 16 Bytes [11]. Ver Figura 3.6.

El número de serie consiste de 5 valores hexadecimales, se podría utilizar esto para hacer una operación dependiendo del número de serie. Las conexiones de los pines dependen del módulo Arduino que se utilice. Ver Tabla 4 [12].

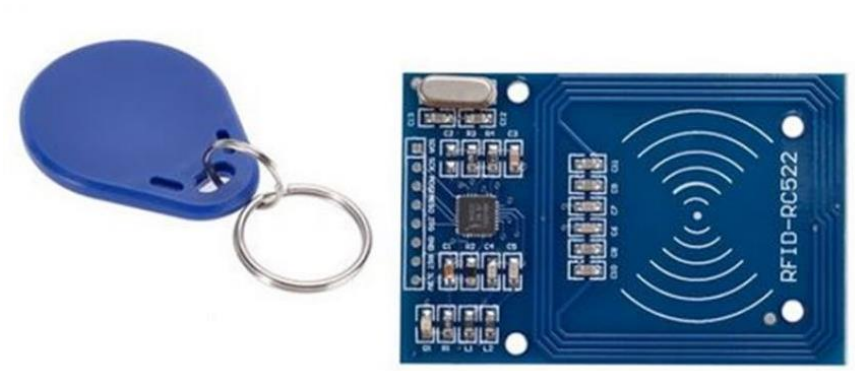

Figura 3.6: Módulo y tarjeta CFRC522.

MÓDULO MFRC522	ARDUINO
SDA	D53
SCK	D52
MOSI	D51
MISO	D50
IRQ	--
GND	GND
RST	D49
Vcc	3.3 V

Tabla 4: Descripción de la conexión de los Pines entre el módulo MFRC522 y Arduino Mega.

3.5 Módulo GPRS SIMCOM SIM900 Quad Band GSM Shield.

El Módulo GPRS SIM900 es un módulo compacto creado para ser compatible con los modelos de Arduino y utilizado para generar una comunicación inalámbrica GPRS usando una red celular permitiendo utilizar llamadas telefónicas, SMS y conexión a internet. Ver Figura 3.7.

La tarjeta está basada en un módulo SIM900 GSM 4, su configuración es a través de comandos AT y controlada por comunicación UART [13]. Ver Tabla 5.

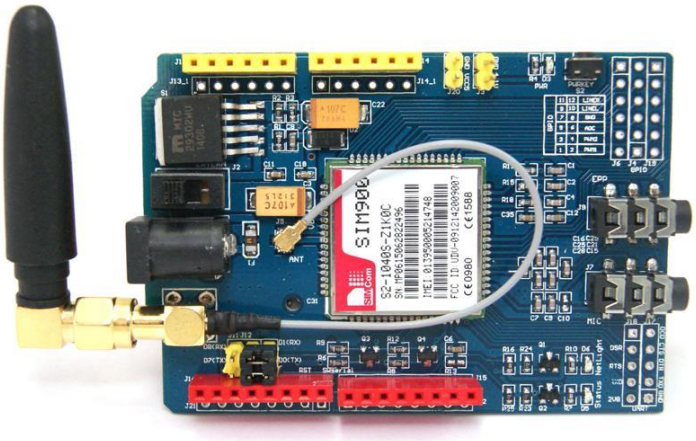

Figura 3.7: Módulo GPRS SIMCOM SIM900 GSM [14].

COMANDO	DESCRIPCIÓN.
AT	Comprueba el estado del módulo.
AT+CPIN=”xxxx”	Introduce el PIN de la tarjeta SIM.
AT+CREG?	Comprueba la conexión a la red.
AT+CGATT=1	Conecta a la red GPRS.
AT+CSTT=”APN”,”xxxx”,”xxx”	Define la APN, usuario y contraseña.
AT+CIICR	Activa el perfil de datos inalámbrico.
AT+CIFSR	Obtiene la IP de la tarjeta SIM.
AT+CIPSTART=”TCP”,”xxx”,”xxx”	Indica el tipo de conexión, dirección IP y puerto a utilizar.
AT+CIPSEND	Prepara el envío de paquetes de datos.
AT+CIPCLOSE	Cierra la conexión.
AT+CIPSHUT	Cierra el contexto PDP del GPRS.

Tabla 5: Comandos AT para el módulo GPRS a internet.

Este módulo posee sus propias especificaciones: cuenta con conexión con puerto serial, compatible con diferentes Quad-Band, entre otras [15]. Ver Tabla 6.

Especificaciones del Módulo GPRS.
Compatible con Arduino.
Conexión con puerto serial.
Quad-Band 850/ 900/ 1800/ 1900 Mhz
GPRS multi-slot clase 10/8
GPRS mobile station clase B
Compatible GSM fase 2/2+
Clase 4 (2 W (AT) 850 / 900 MHz)
Clase 1 (1 W (AT) 1800 / 1900MHz)
TCP/UP embebido
Soporta RTC
Consumo de 1.5 mA (susp)

Tabla 6: Especificaciones del módulo GPRS.

3.6 Implementación del sistema en el controlador Arduino.

A continuación, se muestra un ejemplo del código fuente en la IDE de Arduino de cada dispositivo conectado al controlador, mostrando la forma de conexión física, las funciones utilizadas, el manejo de los datos y la visualización de los mismos.

3.6.1 LCD 16x2 - EW162B0YMY.

La pantalla LCD es el medio de interacción entre el usuario y el controlador. Ver Figura 3.8.

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(32, 30 ,28 , 26, 24, 22);

void setup() {
  lcd.begin(16, 2);

  lcd.print("hello, world!");
  delay(1500);
  lcd.clear();
}

void loop() {
  lcd.setCursor(0, 1);
  lcd.print("hello, again!");
  delay(1000);
  lcd.clear();
}
```

Figura 3.8: Ejemplo del código fuente para el uso del LCD.

El código empieza llamando a la librería *LiquidCrystal.h* que es la encargada de la configuración de los pines a usar en el controlador por medio de la función LiquidCrystal y del manejo de las funciones para la utilización de la LCD. La librería *LiquidCrystal.h* puede ser agregada directamente desde el IDE de Arduino en la sección: *Programa/Incluir Librería/Gestionar Librerías.* Ver Figura 3.9.

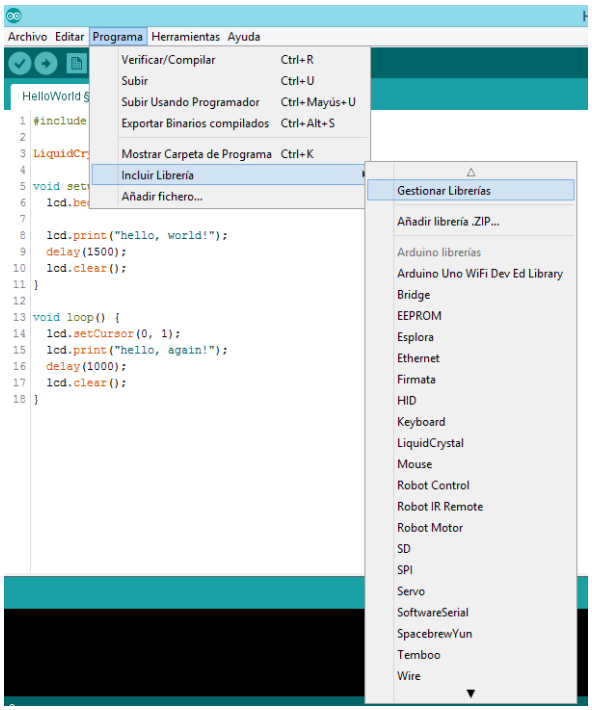

Figura 3.9: Proceso para agregar la librería LiquidCrystal en el IDE Arduino.

La función lcd.begin() utilizada en el void setup indica al controlador el tipo de lcd que se va a usar en este caso una de 16x2 y posiciona al cursor en la casilla de la primera fila y la primera columna. La función lcd.print() indica los caracteres que van a ser mostrados en la lcd, mientras que la función lcd.clear() limpia la pantalla devolviendo al cursor en su posición inicial.

En el void loop la función lcd.cursor() posiciona el cursor en una casilla especificada dentro de la función, en este caso la columna 0 y la fila 1. Ver Figura 3.10.

```
5  void setup() {
6    lcd.begin(16, 2);
7
8    lcd.print("hello, world!");
9    delay(1500);
10   lcd.clear();
11 }
12
13 void loop() {
14   lcd.setCursor(0, 1);
15   lcd.print("hello, again!");
16   delay(1000);
17   lcd.clear();
18 }
```

Figura 3.10: Ejemplos de uso de algunas funciones de la librería LiquidCrystal.

La conexión física entre la LCD y el módulo Arduino se muestra en la Figura 3.11. El brillo de la LCD se regula por un potenciómetro de 10 [KΩ].

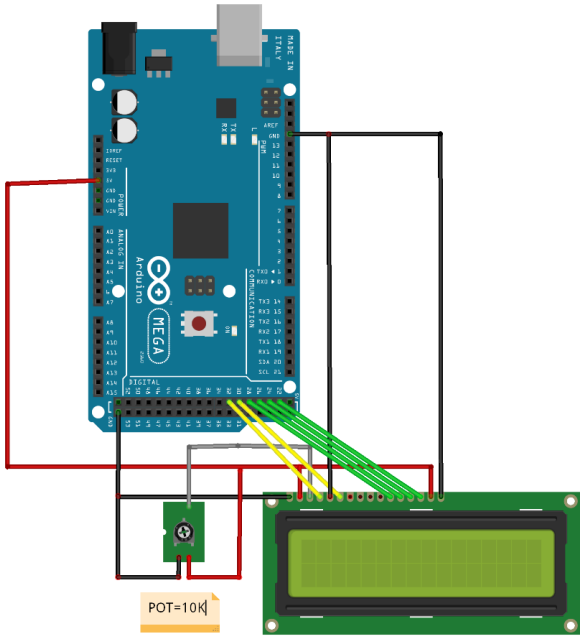

Figura 3.11: Forma de conexión de la LCD 16x2 y el módulo Arduino.

3.6.2 Módulo GPS.

El módulo GPS es el encargado de dar los datos de la velocidad, la posición del vehículo, la fecha y hora en tiempo real. Ver Figura 3.12.

```
#include <TinyGPS.h>//incluimos TinyGPS

TinyGPS gps;//Declaramos el objeto gps
int year;
byte month, day, hour, minute, second, hundredths;

void setup(){
Serial.begin(9600);//Iniciamos el puerto serie
Serial1.begin(9600);//Iniciamos el puerto serie del gps
}

void loop(){
while(Serial3.available()){
int c = Serial3.read();
if(gps.encode(c)){
  float latitude, longitude;
  gps.f_get_position(&latitude, &longitude);
  gps.crack_datetime(&year,&month,&day,&hour,&minute,&second,&hundredths);

    if (hour == 0)
        hour = 24;
    if (hour == 1)
        hour = 25;
    if (hour == 2)
        hour = 26;
    if (hour == 3)
```

```
25          hour = 26;
26      if (hour == 3)
27          hour = 27;
28      if (hour == 4)
29          hour = 28;
30
31  ////////// GPS EN LCD
32      lcd.print("FECHA:");
33      lcd.setCursor(0, 1);
34      lcd.print(day);lcd.print("/");lcd.print(month);lcd.print("/");lcd.print(year);
35      delay(2500);lcd.clear();
36
37      lcd.print("HORA:");
38      lcd.setCursor(0, 1);
39      lcd.print(hour-5);lcd.print(":");lcd.print(minute);lcd.print(":");lcd.print(second);
40      delay(2500);lcd.clear();
41
42      lcd.print("VELOCIDAD:");
43      lcd.setCursor(0, 1);
44      lcd.print(gps.f_speed_kmph());lcd.print(" Km/h");delay(2500);lcd.clear();
45  }}
46  }
```

Figura 3.12: Código fuente para la programación del módulo GPS.

El código fuente empieza llamando a la librería *TinyGPS.h* que es la responsable de interpretar los datos recibidos desde los satélites hacia nuestro GPS (para agregar la librería *TinyGPS.h* al IDE de Arduino se realiza el mismo procedimiento realizado para la librería LiquidCristal. Ver Figura 3.8). Luego procedemos a crear la instancia gps y a crear las variables que necesitamos para mostrar fecha y hora. Ver Figura 3.13.

```
1  #include <TinyGPS.h>//incluimos TinyGPS
2
3  TinyGPS gps;//Declaramos el objeto gps
4  int year;
5  byte month, day, hour, minute, second, hundredths;
6
```

Figura 3.13: Creación de instancias y variable para el uso del módulo GPS.

En el void setup se configura la velocidad de transmisión del módulo GPS con el módulo Arduino, estas velocidades deben ser iguales para que no exista problema con la transmisión de datos por el puerto serial. Para esta configuración se utiliza el Puerto Serial 1 del módulo Arduino, siendo la conexión el Rx del sensor al Rx del Arduino y el Tx del sensor al Tx del Arduino. Ver Figura 3.14.

```
7  void setup(){
8  Serial.begin(9600);//Iniciamos el puerto serie
9  Serial1.begin(9600);//Iniciamos el puerto serie del gps
10 }
```

Figura 3.14: Configuración de la velocidad de transmisión del módulo GPS.

Dentro del void loop se crea un bucle que funciona si y solo si el módulo GPS está recibiendo señal. Luego se procede a crear un valor entero 'c' al que se le asigna los datos recibidos por el módulo GPS en ese momento, los datos recibidos por el módulo GPS son caracteres codificados, la función gps.encode() es la encargada de decodificarlos y administrarlos para su respectivo uso. Ver Figura 3.15.

```
12 void loop(){
13 while(Serial3.available()){
14 int c = Serial3.read();
15 if(gps.encode(c)){
```

Figura 3.15: Proceso de lectura de datos del módulo GPS.

Las funciones gps.f_get_position() y gps.crack_datetime() devuelven las coordenadas geográficas y los datos de fecha y hora en las variables especificadas respectivamente. Ver Figura 3.16.

```
15 if(gps.encode(c)){
16   float latitude, longitude;
17   gps.f_get_position(&latitude, &longitude);
18   gps.crack_datetime(&year,&month,&day,&hour,&minute,&second,&hundredths);
```

Figura 3.16: Actualización de variables según los datos obtenidos desde el módulo GPS.

Procedemos a mostrar los datos obtenidos por el módulo GPS a través de una LCD usando funciones de la librería LiquidCrystal. Ver Figura 3.9. Para obtener la hora según nuestra zona horaria procedemos a restar 5 al valor obtenido en la variable hora. Ver Figura 3.17.

```
31 ////////// GPS EN LCD
32     lcd.print("FECHA:");
33     lcd.setCursor(0, 1);
34     lcd.print(day);lcd.print("/");lcd.print(month);lcd.print("/");lcd.print(year);
35     delay(2500);lcd.clear();
36
37     lcd.print("HORA:");
38     lcd.setCursor(0, 1);
39     lcd.print(hour-5);lcd.print(":");lcd.print(minute);lcd.print(":");lcd.print(second);
40     delay(2500);lcd.clear();
```

Figura 3.17: Visualización de los datos de Fecha y Hora obtenidos por el módulo GPS.

El valor de la velocidad se lo obtiene por medio de la función gps.f_speed_kmph() y sus unidades son en Km/h. Ver Figura 3.18.

```
42     lcd.print("VELOCIDAD:");
43     lcd.setCursor(0, 1);
44     lcd.print(gps.f_speed_kmph());lcd.print(" Km/h");delay(2500);lcd.clear();
```

Figura 3.18: Adquisición del dato de velocidad obtenido por el módulo GPS.

La conexión física entre el módulo GPS y el módulo Arduino son mostradas en la Figura 3.19, las conexiones pueden ser directas entre ambos módulos.

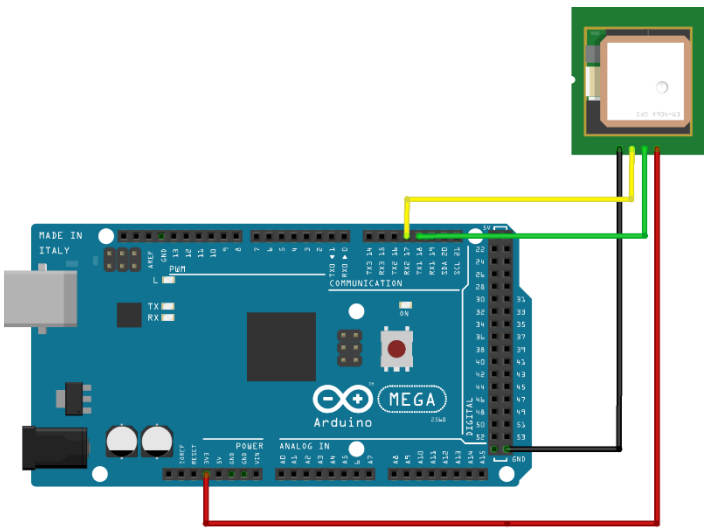

Figura 3.19: Conexión entre el Módulo GPS-6MV2 y Arduino.

3.6.3 Módulo RFID MFRC522 RF.

El código del Módulo RFID MFRC522 RF es mostrado en la Figura 3.20.

```
#include <deprecated.h>
#include <MFRC522Extended.h>
#include <require_cpp11.h>
#include <MFRC522.h>
#include <SPI.h>
#define RST_PIN         49
#define SS_PIN          53
MFRC522 mfrc522(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;
void setup() {
    Serial.begin(9600);
    SPI.begin();
    mfrc522.PCD_Init();
}
void loop() {
    if ( ! mfrc522.PICC_IsNewCardPresent())
        return;
    if ( ! mfrc522.PICC_ReadCardSerial())
        return;
    dump_byte_array(mfrc522.uid.uidByte, mfrc522.uid.size);
    mfrc522.PICC_HaltA();
    mfrc522.PCD_StopCrypto1();}
void dump_byte_array(byte *buffer, byte bufferSize) {
    for (byte i = 0; i < bufferSize; i++) {
        Serial.print(buffer[i] < 0x10 ? "0" : "");
        Serial.print(buffer[i], HEX); }
Serial.println();}
```

Figura 3.20: Código fuente para la programación del Módulo MFRC522.

El código inicia llamando a las librerías *deprecated.h*, *MFRC522Extend.h, require_cpp11.h, MFRC522.h* que son las librerías para la conexión entre el módulo y la tarjeta. La librería *SPI.h* es la encargada de la transmisión de datos entre el módulo MFRC522 y el módulo Arduino. Ver Figura 3.21. En caso de no tenerlas instaladas, se realiza el mismo procedimiento de instalación realizado con la librería LiquidCristal. Ver Figura 3.8.

```
#include <deprecated.h>
#include <MFRC522Extended.h>
#include <require_cpp11.h>
#include <MFRC522.h>
#include <SPI.h>
```

Figura 3.21: Librerías a usar en el Módulo MFRC522.

Definimos los pines a utilizar en la comunicación SPI con el Arduino, creamos la instancia mfrc522 y le agregamos la clave a usar. Ver Figura 3.22.

```
6  #define RST_PIN         49
7  #define SS_PIN          53
8  MFRC522 mfrc522(SS_PIN, RST_PIN);
9  MFRC522::MIFARE_Key key;
```

Figura 3.22: Configuración de los pines de comunicación entre el Módulo MFRC522 y Arduino y la clave a usar.

En el void setup se inicializan ambos módulos, al ser conexión SPI no es necesario establecer la velocidad en el módulo MFRC522. Ver Figura 3.23.

```
10 void setup() {
11     Serial.begin(9600);
12     SPI.begin();
13     mfrc522.PCD_Init();
14 }
```

Figura 3.23: Inicialización de los módulos Arduino y MFRC522.

En el void loop se empieza a leer la tarjeta, de no encontrarla vuelve a esperar hasta que encuentre una. Ver Figura 3.24.

```
15 void loop() {
16     if ( ! mfrc522.PICC_IsNewCardPresent())
17         return;
18     if ( ! mfrc522.PICC_ReadCardSerial())
19         return;
```

Figura 3.24: Espera por la presencia de una tarjeta.

Finalmente, cuando encuentra una tarjeta lee su contenido mostrándolo por el puerto serial. Ver Figura 3.25.

```
20      dump_byte_array(mfrc522.uid.uidByte, mfrc522.uid.size);
21      mfrc522.PICC_HaltA();
22      mfrc522.PCD_StopCrypto1();}
23  void dump_byte_array(byte *buffer, byte bufferSize) {
24      for (byte i = 0; i < bufferSize; i++) {
25          Serial.print(buffer[i] < 0x10 ? "0" : "");
26          Serial.print(buffer[i], HEX); }
27  Serial.println();}
```

Figura 3.25: Proceso de lectura del contenido de la tarjeta.

La conexión entre e módulo MFRC522 y el módulo Arduino es mostrado en la Figura 3.26. Ambos módulos pueden ser conectados directamente.

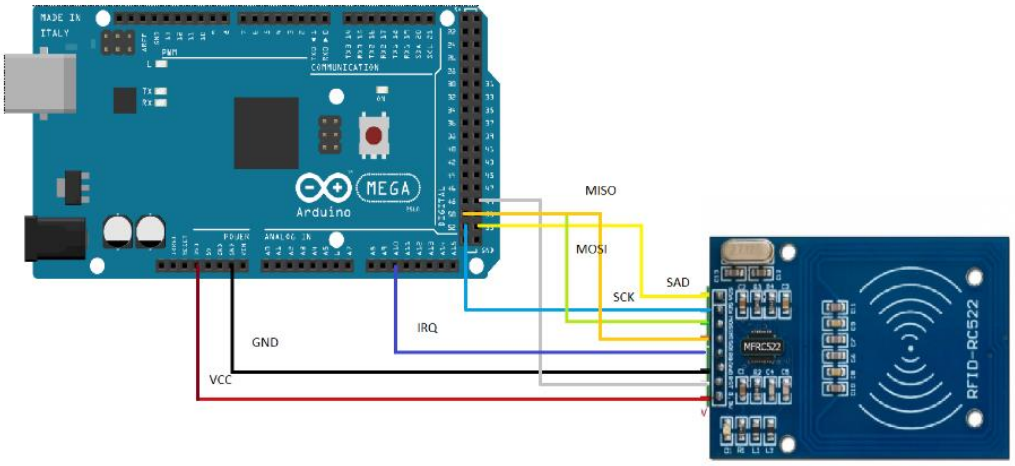

Figura 3.26: Conexión física entre el módulo MFRC522 Y Arduino Mega.

3.6.4 Botoneras y Leds.

Los pines de entrada para los cambios de estado serán los pines pull-up D20 y D21 del módulo Arduino. Ver Figura 3.27.

```
64  const byte inAccidente = 20;
65  const byte inRobo = 21;
66
67  const byte ledNube = 47;
68  const byte ledOKgps = 46;
69  const byte ledAux = 48;
```

Figura 3.27: Definición de los pines de entrada para uso de las botoneras.

La conexión física de las botoneras al módulo Arduino se muestra en la Figura 3.28.

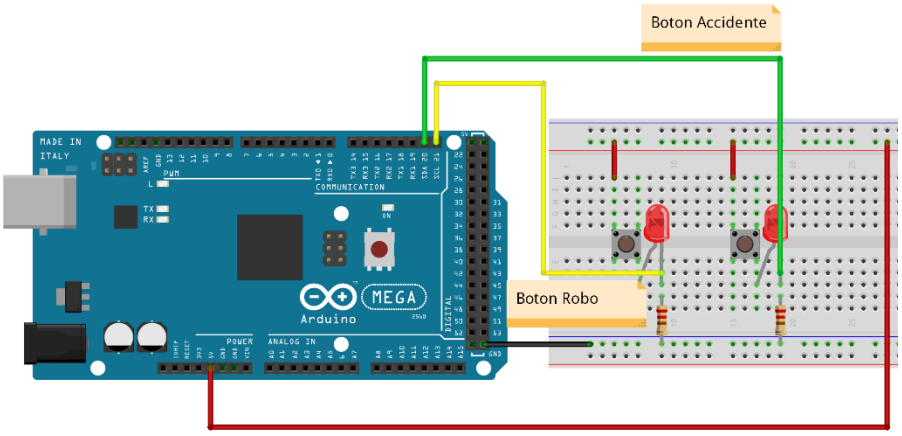

Figura 3.28: Conexión física entre el módulo Arduino y las botoneras.

3.6.5 Módulo GPRS.

La configuración del módulo GPRS es a través de comandos AT. El código para petición web es la siguiente:

El programa empieza llamando la librería *SoftwareSerial.h*, esta librería ya está instalada por defecto en el IDE de Arduino. A continuación, especificamos los pines que usaremos para el software serial que serán los pines 7 y 8. El software serial permite emular un puerto serial sin necesidad de usar un puerto real, basta con crear una instancia una vez declarado el mismo en este caso es “esp”. Ver Figura 3.29.

Finalmente creamos constantes con formato JSON que son utilizados por la API de Ubidots el almacenamiento de datos.

```
21
22  // PARAMETROS -- GPRS
23  #include <SoftwareSerial.h>//incluimos SoftwareSerial
24  SoftwareSerial esp(7,8);// RX, TX
25
26  String posicion = "1";//////////////////////////////////String Velocidad
27
28  String robo = "1"; //1
29
30  String id_tarjeta = "187";////////////////////////////// String id_tarjeta
31  String nombre = "\"Paul\"";
32  String apellidos = "\"Aguilar\"";
33
34  String Post = "POST /api/v1.6/variables/";
35
36  String id_usuario = "5984d2c9c03f9773bdda9064";
37  String id_velocidad = "5982c385c03f974499a02ff5";
38  String id_posicion = "5983c2fbc03f9758bc4ac5a8";
39  String id_aux = "59835ac8c03f976fe44f5736";
40
41  String id_f = "/values HTTP/1.1";
42  String token = "X-Auth-Token: AkRSIDIswuwUw4gxldtRexwZtB1DjO";
43  String Host = "Host: things.ubidots.com";
44  String cc = "Connection: close";
45  String ct = "Content-Type: application/json";
46  String cl = "Content-Length: ";
```

Figura 3.29: Creación de parámetros con formato JSON para la comunicación con la API de Ubidots.

En el void setup procedemos a conectar a la red celular, para que el módulo tenga conexión a red hay que enviar el comando AT+CPIN=”1111” (“1111” es el pin de la tarjeta SIM, este número depende de la operadora celular), además es necesario esperar un tiempo de 25 segundos aproximadamente para asegurar una correcta conexión. Ver Figura 3.30.

```
71 void setup() {
72
73   lcd.begin(16, 2);
74   lcd.print("INICIALIZANDO ");
75   lcd.setCursor(0, 1);
76   lcd.print("CONTROLADOR..");
77   delay(2500);lcd.clear();
78
79   lcd.begin(16, 2);
80   lcd.print("CONECTANDO A");
81   lcd.setCursor(0, 1);
82   lcd.print("RED..");
83
84   esp.println("AT+CPIN=\"1111\""); //Comando AT para introducir el PIN de la tar
85   delay(25000); //Tiempo para que encuentre una RED
86   lcd.clear();
87 }////////////////////////////////////////////////////------ VOID LECTOR()
```

Figura 3.30: Conexión de la tarjeta SIM a la red celular.

En el void loop se iniciará el proceso de petición POST que es el que permite enviar y guardar los datos en la nube. Ver Figura 3.31. Para poder enviar la petición POST y conectar a la red celular hay que configurar el módulo GPRS de la siguiente manera:

1. Enviar el comando AT+CGATT=1; que es el comando que permite iniciar el servicio de envío de paquetes por red.
2. Enviar el comando AT+CSTT=”internet.claro.com.ec”,””,””; que sirve para la configuración APAN (Access Point Name), nombre de usuario y contraseña, estos datos pueden variar dependiendo de la operadora móvil.
3. Enviar el comando AT+CIICR; que es el que activa el sistema GPRS para envío de datos.
4. Enviar el comando AT+CIFSR; que es el que devuelve la dirección IP del módulo.
5. Enviar el comando AT+CIPSTART=”TCP”,”things.ubidots.com”, ”8080”; que es el comando que inicia una conexión TCP (Protocolo de Transmisión de Control) especificando la dirección del servidor y el puerto a usar.
6. Enviar el comando AT+CIPSEND=”” con la petición POST para poder acceder al servidor establecido previamente.

7. Enviar el comando AT+CIPCLOSE para cerrar la conexión TCP establecida en el paso 5.

```
21 void loop()
22 {
23   esp.println("AT+CGATT=1");
24   delay(500);
25
26   esp.println("AT+CSTT=\"internet.claro.com.ec\",\"\",\"\"");
27   delay(500);
28
29   esp.println("AT+CIICR");
30   delay(2000);
31
32   esp.println("AT+CIFSR");
33   delay(2000);
34
35   esp.println("AT+CIPSTART=\"TCP\",\"things.ubidots.com\",\"8080\"");
36   delay(2000);
37
38   esp.print("AT+CIPSEND=");esp.println(usuario.length());
39   delay(1100);
40
41   esp.println(usuario);
42   delay(25000);
43
44   esp.println("AT+CIPCLOSE");
45   delay(1000);
46
```

Figura 3.31: Configuración del módulo GPRS para el envío de peticiones HTTP.

CAPÍTULO 4

4. IMPLEMENTACIÓN DEL SISTEMA.

4.1 Adquisición de datos.

El proceso de adquisición de datos empieza con la conexión del controlador a la red celular a través del módulo GPRS. Ver Figura 4.1.

Figura 4.1: Conexión del controlador a la red celular.

Una vez conectado a la red celular, el sistema espera por la identificación del usuario por medio del lector de tarjeta RFID. Ver Figura 4.2.

Figura 4.2: Módulo RFID esperando validar usuario.

El sistema de reconocimiento está hecho de tal manera que solo un usuario use un controlador en específico, de esta manera no se permitirá a otros usuarios ingresar al sistema. Ver Figura 4.3.

Una vez validado el usuario el sistema envía sus datos a la aplicación web, especificando su: nombre, apellido e identificación de tarjeta. Ver Figura 4.4.

Figura 4.3: Usuario aceptado por el módulo RFID.

Figura 4.4: Envío de paquete de datos de usuario a la base de datos.

Al pasar la validación de usuario, el módulo GPS proporciona una adquisición de datos constante en tiempo real, entre ellos velocidad, fecha y hora. Ver Figuras 4.5, 4.6 y 4.7 respectivamente.

Figura 4.5: Adquisición de datos de velocidad a través del módulo GPS.

Figura 4.6: Adquisición de datos de fecha a través del módulo GPS.

Figura 4.7: Adquisición de datos de hora a través del módulo GPS.

Los datos de posición son mostrados de manera directa en la aplicación web y son enviados al mismo tiempo que los datos de velocidad. Ver Figura 4.8.

El envío del estado de las botoneras interrumpe al sistema y puede ser ejecutado en cualquier momento que sea necesario.

Figura 4.8: Envío de señal de auxilio a la base de datos.

4.2 Envío y recepción de datos.

4.2.1 Ubidots.

Para el envío de datos por medio del módulo GPRS usamos el formato JSON para el empaquetamiento de los datos. Los datos son enviados a la nube de Ubidots para el almacenamiento y la creación de iframe's que serán insertados en la base de datos SQL.

Para enviar los datos hay que tener en cuenta el valor del Token y el ID de la variable. El valor del token es el siguiente: AkRSIDTswuwUw4gxldtRexwZtB1DjO.

Los datos de identificación de usuario son los primeros en ser enviados a la nube debido a que el sistema no empieza si no hay algún usuario reconocido por medio del módulo RFID. El valor del ID de la variable usuario es el siguiente: 5984d2c9c03f9773bdda9064. Ver Figura 4.9.

```
CONEXION A SERVER
POST /api/v1.6/variables/5984d2c9c03f9773bdda9064/values HTTP/1.1
X-Auth-Token: AkRSIDTswuwUw4gxldtRexwZtB1DjO
Host: things.ubidots.com
Connection: close
Content-Type: application/json
Content-Length: 66

{"value":187,"context":{"nombre":"Cristian","apellido":"Pacalla"}}

PETICION ENVIADA
```

Figura 4.9: Envío de datos de identificación de usuario con formato JSON.

Por motivos de costos de red, los datos de velocidad y posición adquiridos por el módulo GPS están programados para su envío cada 3 minutos. Los valores del ID de la variable velocidad y la variable posición son los siguientes respectivamente: 5982c385c03f974499a02ff5 y 5983c2fbc03f9758bc4ac5a8. Ver Figuras 4.10 y 4.11 respectivamente.

```
CONEXION A SERVER
POST /api/v1.6/variables/5982c385c03f974499a02ff5/values HTTP/1.1
X-Auth-Token: AkRSIDTswuwUw4gxldtRexwZtB1DjO
Host: things.ubidots.com
Connection: close
Content-Type: application/json
Content-Length: 16

{"value":0.13}
```

Figura 4.10: Envío de datos de velocidad con formato JSON.

```
CONEXION A SERVER
POST /api/v1.6/variables/5983c2fbc03f9758bc4ac5a8/values HTTP/1.1
X-Auth-Token: AkRSIDTswuwUw4gxldtRexwZtB1DjO
Host: things.ubidots.com
Connection: close
Content-Type: application/json
Content-Length: 58

{"value":1,"context":{"lat":-2.1453891,"lng":-79.9661940}}

PETICION ENVIADA
```

Figura 4.11: Envío de datos de posición con formato JSON.

El envío de las señales de auxilio se realiza cada vez que estas son activadas. Tener en cuenta que se envía una señal a la vez. El ID de la variable estado es el siguiente: 59835ac8c03f976fe44f5736. Ver Figura 4.12.

```
PREPARANDO ENVIO SEÑAL

POST /api/v1.6/variables/59835ac8c03f976fe44f5736/values HTTP/1.1
X-Auth-Token: AkRSIDTswuwUw4gxldtRexwZtB1DjO
Host: things.ubidots.com
Connection: close
Content-Type: application/json
Content-Length: 11

{"value":1}
```

Figura 4.12: Envío de señal de auxilio con formato JSON.

Ubidots almacena los datos recibidos en variables detallando con la hora de envío. Ver Figura 4.13.

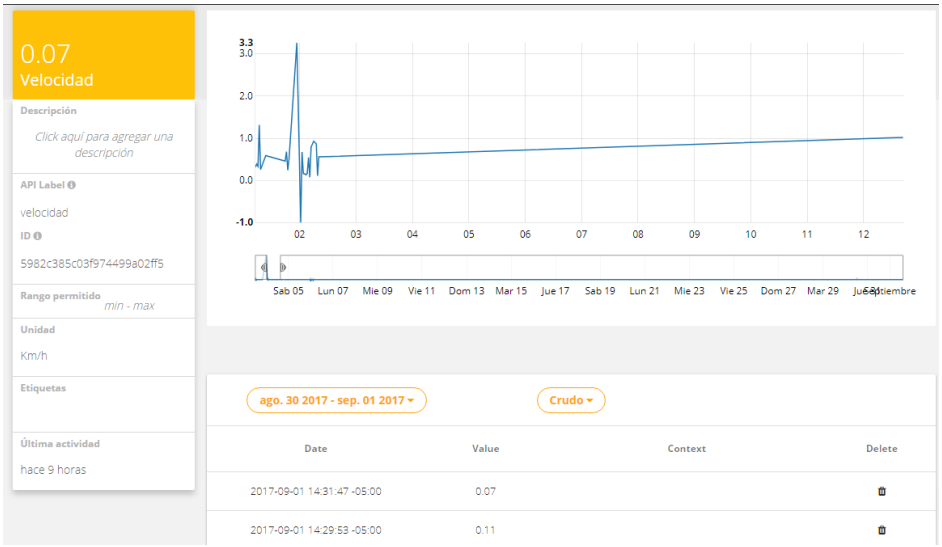

Figura 4.13: Recepción de datos en la variable velocidad en la nube de Ubidots.

Cada variable puede generar uno o más iframe's para su visualización dentro del dashboard de Ubidots. Estos iframe's van a ser insertados en el servidor web SQL y en la aplicación Android. Ver Figura 4.14.

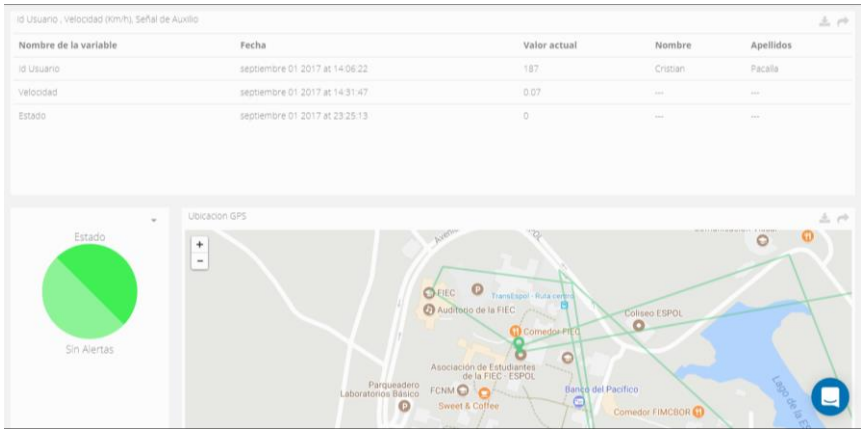

Figura 4.14: Dashboard de la nube Ubidots.

4.2.2 Base de datos y Página web.

Los campos de la tabla usuario de la base de datos son los siguientes: id (identificación), número de tarjeta, nombres, apellidos, ci (cédula de identidad), sexo, fecha de registro a la base de datos, verificar (acceso administrador), estado de la señal del bus, ubicación, contraseña, cooperativa y número de bus. Ver Figura 4.15.

id	tarjeta	nombres	apellidos	ci	direccion	sexo	fechaderegistro	verificar	estadobus	ubicacion	password	cooperativa	nbus
7	456434	Cristhian Marcelo	Pacalla Guaman	925021008	Urdesa Central calle 1era y Guayacanes	Masculino	2017-08-21 15:55:24	NO AUTORIZADO	<iframe width="300" height="260" frameborder="0" s...	<iframe width="100%" height="100%" frameborder="0"...	1234	54	23

Figura 4.15: Visualización de información del usuario por medio de la base de datos.

La tabla usuario guarda toda la información necesaria del conductor, la cooperativa y el bus, además guarda su ubicación y el estado de la señal de auxilio para saber cuándo brindar la ayuda necesaria.

Elegimos SQL ante otras opciones de bases de datos como ORACLE y PARADOX db, por su fácil manejo e instalación, puesto que SQL permite usar una computadora como un servidor.

El sitio web fue realizado con lenguaje de programación HTML para la interfaz y PHP para la conexión a la base de datos. Ver Figura 4.16.

Figura 4.16: Página web HTML para el acceso a la base de datos.

4.3 Métodos de visualización.

Toda información ingresada a la base de datos es confidencial y únicamente el administrador tiene acceso a ella, del mismo modo solo él puede modificar, eliminar, agregar o consultar los campos de las tablas. La dirección de la aplicación web es la siguiente: http://tesisdatos.000webhostapp.com. Ver Figura 4.16.

4.3.1 Página Web.

Para el ingreso al sistema de control de datos en la página web, el usuario debe estar registrado en la base de datos administrador. Ver Figura 4.17.

id	usuario	contrasena
1	cpacalla	12345

Figura 4.17: Tabla administrador en la base de datos.

Al ingresar a la aplicación, dependiendo de su necesidad el usuario dispone de opciones mostradas a través de un menú. Ver Figura 4.18.

Figura 4.18: Menú de la aplicación web.

El usuario posee la capacidad de ingresar o modificar campos de la tabla usuario. Ver Figura 4.19.

Figura 4.19: Ingreso de usuario desde servidor web.

Lo campos presentados deben ser llenados de forma obligatoria debido a que proporcionan la información necesaria del conductor.

Se puede buscar un usuario y modificar sus datos por medio de su cédula de identidad. Ver Figuras 4.20 y 4.21 respectivamente.

Figura 4.20: Consulta de usuarios por su cédula de identidad.

Figura 4.21: Panel para la modificación de la información de usuario.

Nos basamos en la búsqueda por número de Cedula debido a que este número es único para cada usuario, es decir no puede repetirse a diferencia de nombres o apellidos que pueden coincidir con otro usuario y causar confusión.

Los parámetros en la modificación de los capos de usuario también incluyen el número de su tarjeta de acceso, debido a posibles pérdidas o daños de la misma.

La opción consultar en el menú principal ver Figura 4.18, muestra los usuarios registrados en la tabla usuario con sus respectivos campos. Ver Figura 4.22.

Figura 4.22: Visualización de los campos en la tabla usuario.

El color del campo “ESTADO DEL VEHÍCULO” indica el tipo de señal recibida desde el controlador. Al existir una señal de auxilio el administrador es capaz de saber la posición del vehículo haciendo clic en el botón “mapa”, lo que lo conducirá a un panel mostrando una tabla con la identificación del usuario (chofer), la velocidad del vehículo, el indicador del estado de la señal y la ubicación del vehículo en el mapa. Ver Figura 4.23.

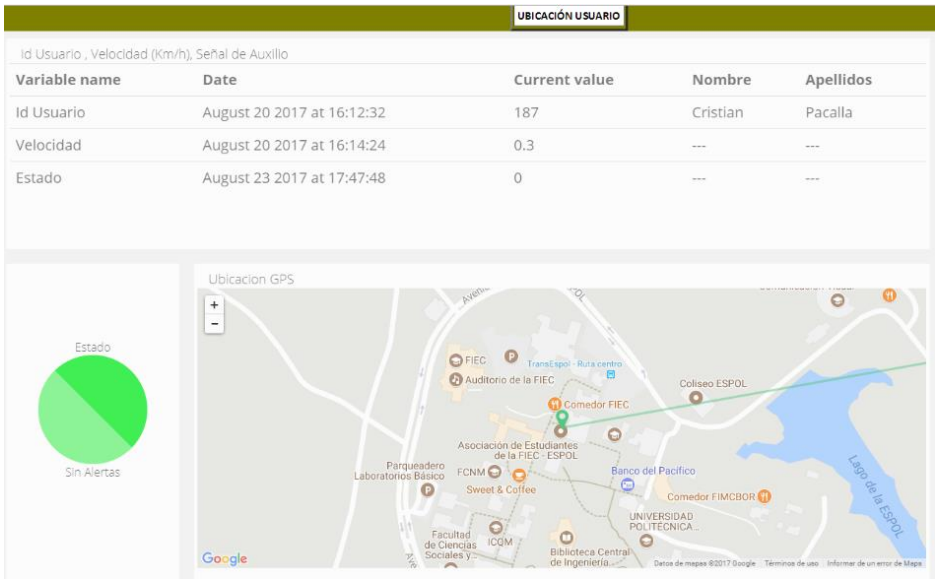

Figura 4.23: Panel de ubicación del conductor.

4.3.2 MyBus (Aplicación móvil Android).

MyBus es una aplicación creada en la plataforma Android y está diseñada para mostrar la información guardada en la base de datos por medio de dispositivos móviles. El acceso a la aplicación es de uso exclusivo del usuario (choferes) registrados en la base de datos.

El ingreso a la aplicación es con la cédula de identidad y una contraseña para los usuarios registrados en la base de datos. Ver Figura 4.24.

Figura 4.24: Ingreso a la aplicación MyBus.

El panel de usuario (chofer) muestra su posición y el estado de señal de auxilio. El uso de la aplicación es de uso opcional para el usuario (chofer). Ver Figura 4.25

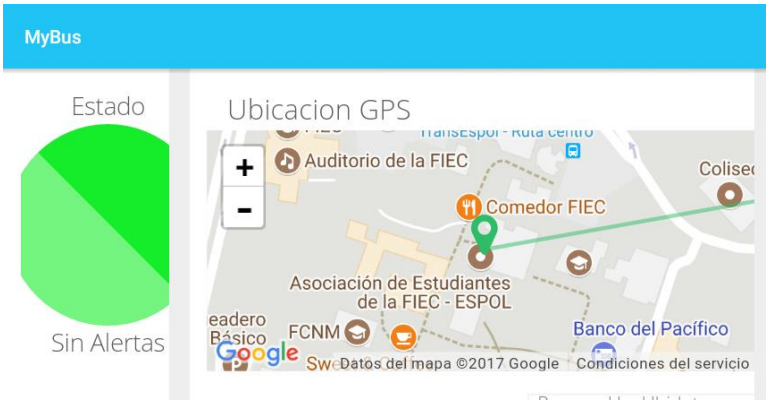

Figura 4.25: Panel Usuario dentro de la aplicación móvil.

4.4 Resultados obtenidos.

La solución propuesta da una solución al problema planteado por este proyecto, debido a que el tiempo que tarda en enviar el controlador cada señal de auxilio es aproximadamente 12 segundos, es decir que en 15 segundos luego de ser enviada una señal de auxilio, un operario de la aplicación web tendrá la responsabilidad de enviar la ayuda correspondiente a la señal recibida al lugar que sea necesario. Ver Figuras 4.26 y 4.27 respectivamente.

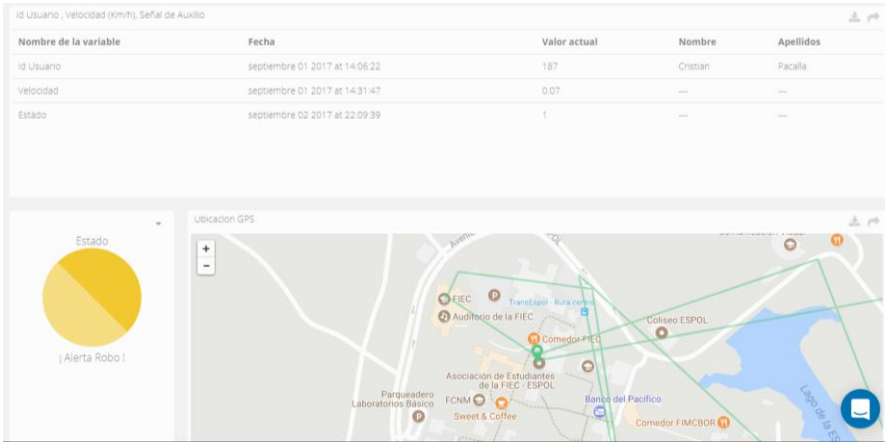

Figura 4.26: Recepción de la señal de robo.

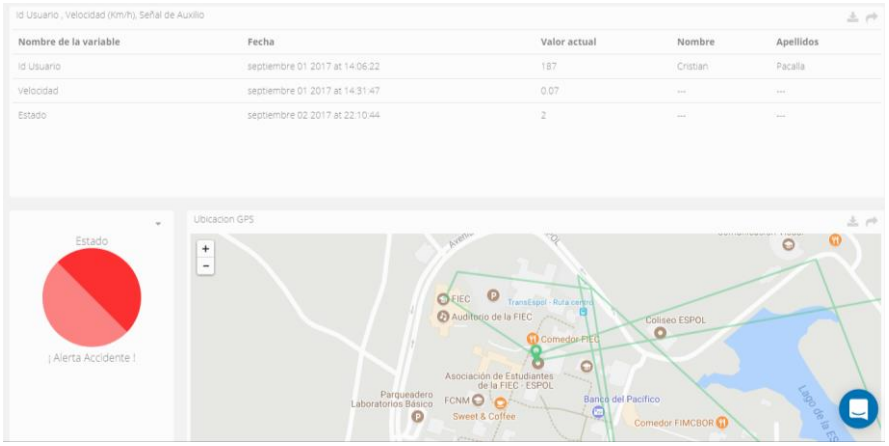

Figura 4.27: Recepción de la señal de accidente.

El tiempo de envío de 12 segundos es por motivos de espera para que los comandos AT sean enviados correctamente y el módulo GPRS pueda tener un buen funcionamiento.

Estos 12 segundos es la demora fuera de la “hora pico” en la transmisión de datos, para este momento del día dependiendo del tráfico de datos el tiempo de envío puede aumentar entre 15 y 20 segundos, lo que de todas maneras en aceptable.

Con esta solución esperamos incrementar la confianza de los usuarios externos, disminuir las muertes por accidentes vehiculares y disminuir el índice delictivo en el transporte el público.

CONCLUSIONES Y RECOMENDACIONES

Con el desarrollo y la implementación de este sistema se pudo disminuir notoriamente el tiempo de auxilio en problemas como accidentes o robos, debido al rápido envío de datos a través de la red celular y la amigable interfaz del usuario para saber qué tipo de ayuda debe enviar y a qué lugar debe de ir.

El almacenamiento de datos en la nube favorece enormemente a la visualización de los datos, permitiendo compartir los en aplicaciones externas cómo, por ejemplo: aplicaciones móviles.

La aplicación de este sistema permitirá un mejor nivel de seguridad y una mejora en el tráfico vehicular, esto incrementará la confianza de los usuarios debido a que se podrá enviar la ayuda necesaria a graves problemas cómo accidentes violentos y la rápida identificación del responsable del vehículo de transporte público (chofer) en caso de que él se dé a la fuga siendo el causante del accidente.

Con la implementación de este sistema en vehículos se podrá crear constancias estadísticas que permitirán conocer los lugares que necesiten más resguardo policial o mejoras viales y de esta forma fomentar una mejor educación vial en el Ecuador.

La adquisición de datos del módulo GPS es constante una vez se haya establecido la conexión con el satélite, pero es susceptible a perderse al momento de obstruir la línea de vista con materiales de gran densidad, por ejemplo: hormigón.

Para una correcta comunicación del módulo GPRS al servidor, la transmisión tarda 25 segundos aproximadamente debido a que cada comando AT demora cerca de un segundo en ser enviado.

Hay que tener en cuenta el tráfico de datos externo, esto puede saturar la red celular y ralentizar el envío de datos hacia el servidor.

El uso de una aplicación móvil en la implementación de cualquier proyecto lo vuelve más amigable al usuario, permitiéndole tener un mejor control sobre el sistema que se esté usando sin necesidad de usar un dispositivo fijo.

Para una lectura correcta del módulo RFID, la posición de la tarjeta debe ser menor a 3 cm, de otro modo está no podrá ser leída.

Para una correcta conexión a la red celular el módulo GPRS debe esperar 25 segundos aproximadamente para no tener problemas con la transmisión de datos.

La velocidad del envío de paquetes de datos depende directamente del nivel de señal celular de la operadora a usar. En caso de aplicar este sistema a vehículos interprovinciales es necesario que las carreteras tengan una buena cobertura de red.

Para el envío de datos al servidor web es necesario activar el envío de datos por red a través de los comandos at y de la misma forma cerrar la conexión una vez enviados los datos, de esta forma evitamos un consumo de datos innecesario. Ver tabla 5.

En el formato JSON el paquete a enviar debe especificar la longitud de la petición de otra forma no será enviados.

El servidor tiene un mejor rendimiento cuando su velocidad de conexión es alta, de esta manera actualiza de mejor manera los datos recibidos.

Existen varias versiones del módulo GPRS que en su operan en bandas específicas con su país de fabricación, al momento de adquirir uno hay que asegurarse que el módulo trabaje dentro de las frecuencias establecidas en el país a ser implementado.

Obtener una base de datos de respaldo en caso de que nuestro servidor por circunstancias externas tenga perdidas de información o a su vez se elimine por completo.

Se recomienda usar Android Studio en una computadora con alto grado de procesamiento o una computadora con pocos procesos activos, debido a que este programa consume demasiados recursos y puede afectar el rendimiento de la misma.

BIBLIOGRAFÍA

[1] Agencia Nacional de Tránsito, (2017, abril). Estadísticas de transporte terrestre y seguridad vial [online]. Disponible en: http://www.ant.gob.ec/index.php/descargable/file/4024-siniestros-marzo-2017

[2] Rouse Margaret, (2015, enero). SQL o lenguaje de consultas estructuradas [online]. Disponible en: http://searchdatacenter.techtarget.com/es/definicion/SQL-o-lenguaje-de-consultas-estructuradas

[3] Wikipedia, (2017, agosto). SQL [online]. Disponible en: https://es.wikipedia.org/wiki/SQL

[4] Ubidots, (2014). Internet de las Cosas con Ubidots [online]. Disponible en: https://ubidots.com/docs/es/get_started/introduccion.html

[5] MCI Electronics. Arduino Mega 2560 R3 [online]. Disponible en: http://arduino.cl/arduino-mega-2560/

[6] Panamahitek, (2013, enero). Arduino Mega: Características, Capacidades y donde conseguirlo en Panamá [online]. Disponible en: http://panamahitek.com/arduino-mega-caracteristicas-capacidades-y-donde-conseguirlo-en-panama/

[7] Yepez Jovanna, (2017, marzo). Características de los Pines de la LCD 16x2 [online]. Disponible en: https://jovannayepez.wordpress.com/2014/03/24/caracteristicas-de-los-pines-de-la-lcd-16x2/

[8] Electrotec, (2017). Ilustración de la LCD 16x2 [online]. Disponible en: http://electrotec.pe/tienda/lcd-16x2

[9] Naylampmechatronics, (2016, julio). Tutorial Módulo GPS con Arduino [online]. Disponible en http://www.naylampmechatronics.com/blog/18_Tutorial-Módulo-GPS-con-Arduino.html

[10] Naylampmechatronics, (2016, julio). Ilustración del Módulo GPS [online]. Disponible en http://www.naylampmechatronics.com/blog/18_Tutorial-Módulo-GPS-con-Arduino.html

[11] Hetpro, (2016, junio). Módulo Lector RC522 RFID RF con Arduino [online]. Disponible en: https://hetpro-store.com/TUTORIALES/rc522-rfid-escritura-lectura/

[12] Naylampmechatronics, (2016, julio). Tutorial módulo Lector RFID RC522 [online]. Disponible en: http://www.naylampmechatronics.com/blog/22_Tutorial-Lector-RFID-RC522.html

[13] Prometec, (2017). Comandos AT para GSM, GPRS y GPS [online]. Disponible en: https://www.prometec.net/comandos-at-gsm-gprs-gps/#

[14] Hetpro, (2015, octubre). Ilustración del Módulo GPRS SIMCOM SIM900 GSM [online]. Disponible en: https://hetpro-store.com/TUTORIALES/sim900-gsm-shieldarduino/

[15] Botscience, (2013). Shield GPRS/GSM (Celular) SIM900 para Arduino [online]. Disponible en: http://botscience.net/store/index.php?route=product/product&product_id=71

ANEXOS

Código fuente usado en el controlador Arduino:

```
// IDENTIFICADOR TARJETA

 #include <deprecated.h>
 #include <MFRC522Extended.h>
 #include <require_cpp11.h>
 #include <MFRC522.h>
 #include <SPI.h>
 #define RST_PIN         49
 #define SS_PIN          53
 MFRC522 mfrc522(SS_PIN, RST_PIN);
 MFRC522::MIFARE_Key key;

 String codigo;
 // LIBRERIAS
 #include <LiquidCrystal.h>//LCD
 #include <SoftwareSerial.h>//incluimos SoftwareSerial
 #include <TinyGPS.h>//incluimos TinyGPS

 #define IP "things.ubidots.com" // UBIDOTS

 // PARAMETROS -- GPRS
 SoftwareSerial esp(7,8);// RX, TX

 String posicion = "1";//////////////////////////////////String Velocidad

 String robo = "1"; //1
```

```

 String id_tarjeta = "187";////////////////////////////// String id_tarjeta
 String nombre = "\"Cristian\"";
 String apellidos = "\"Pacalla\"";

 String Post = "POST /api/v1.6/variables/"; //Campo de temperatura

 String id_usuario = "5984d2c9c03f9773bdda9064";
 String id_velocidad = "5982c385c03f974499a02ff5";
 String id_posicion = "5983c2fbc03f9758bc4ac5a8";
 String id_aux = "59835ac8c03f976fe44f5736";

 String id_f = "/values HTTP/1.1";
 String token = "X-Auth-Token: AkRSIDTswuwUw4gxldtRexwZtB1DjO";
 String Host = "Host: things.ubidots.com";
 String cc = "Connection: close";
 String ct = "Content-Type: application/json";
 String cl = "Content-Length: ";

 //LCD
 LiquidCrystal lcd(32, 30 ,28 , 26, 24, 22);

 // GPS
 TinyGPS gps;//Declaramos el objeto gps---RX A TX3

 int year;
 int lector = 0;
```

```
  byte month, day, hour, minute, second, hundredths;
  unsigned long chars;
  unsigned short sentences, failed_checksum;

  //BOTONES -- INTERRUPCIONES -- LED'S OK

  int tiempo = 0;

  const byte PULSO = 9;

  const byte inAccidente = 20;
  const byte inRobo = 21;

  const byte ledOk = 41;
  const byte ledAux2 = 40;
  const byte ledAux1 = 43;
void setup() {
  ////////////////////////////////////////////////--LECTOR TARJETA
  SPI.begin();
  mfrc522.PCD_Init();
  ////////////////////////////////////////////////-- GPS
  Serial.begin(9600);//Iniciamos el puerto serie
  Serial1.begin(9600);//Iniciamos el puerto serie del gps
  esp.begin(19200);
//  pinMode(ledAux, OUTPUT);  pinMode(PULSO, OUTPUT);
  ////////////////////////////////////////////////-- LCD
  delay(2500);
```

```
  lcd.begin(16, 2);
    digitalWrite(PULSO, HIGH);
  lcd.print("INICIALIZANDO ");
  lcd.setCursor(0, 1);
  lcd.print("CONTROLADOR..");
  delay(3000);lcd.clear();
      digitalWrite(PULSO, LOW);

  lcd.begin(16, 2);

  lcd.print("CONECTANDO A");
  lcd.setCursor(0, 1);
  lcd.print("RED..");

  esp.println("AT+CPIN=\"1111\""); //Comando AT para introducir el PIN de la tarjeta
  delay(25000); //Tiempo para que encuentre una RED
  Serial.println("CONECTADO A RED");
  lcd.clear();

  pinMode(ledOk, OUTPUT);
  pinMode(ledAux1, OUTPUT);
  pinMode(ledAux2, OUTPUT);

  pinMode(inAccidente, INPUT_PULLUP);
  pinMode(inRobo, INPUT_PULLUP);
}////////////////////////////////////////////////////////////------ VOID LECTOR()

```

```
void Lector(){
 if ( ! mfrc522.PICC_IsNewCardPresent())
        return;
    if ( ! mfrc522.PICC_ReadCardSerial())
        return;
    dump_byte_array(mfrc522.uid.uidByte, mfrc522.uid.size);
    mfrc522.PICC_HaltA();
    mfrc522.PCD_StopCrypto1();
}
void dump_byte_array(byte *buffer, byte bufferSize) {
    for (byte i = 0; i < bufferSize; i++) {
        Serial.print(buffer[i] < 0x10 ? "0" : "");
        Serial.print(buffer[i], HEX);
        codigo = buffer[i];
        }
    Serial.println();
    Serial.println(codigo);
}////////////////////////////////////////////////////////////----- VOID GPS

void GPS(){
digitalWrite(led0k, LOW);
auxilio(robo,id_aux);

while(Serial1.available()){
int c = Serial1.read();
if(gps.encode(c)){

```

```

  auxilio(robo,id_aux);

  digitalWrite(led0k, HIGH);
  float latitude, longitude;
  float Position;

  auxilio(robo,id_aux);

  gps.f_get_position(&latitude, &longitude);
  gps.crack_datetime(&year,&month,&day,&hour,&minute,&second,&hundredths);

    if (hour == 0)
        hour = 24;
    if (hour == 1)
        hour = 25;
    if (hour == 2)
        hour = 26;
    if (hour == 3)
        hour = 27;
    if (hour == 4)
        hour = 28;

////////// GPS EN LCD Y ENVIO DE DATOS

    auxilio(robo,id_aux);

```

```
    lcd.print("FECHA:");
    lcd.setCursor(0, 1);
    lcd.print(day);lcd.print("/");lcd.print(month);lcd.print("/");lcd.print(year);
    delay(2500);

    auxilio(robo,id_aux);

    lcd.clear();

    lcd.print("HORA:");
    lcd.setCursor(0, 1);
    lcd.print(hour-5);lcd.print(":");lcd.print(minute);lcd.print(":");lcd.print(second);

    auxilio(robo,id_aux);

    delay(2500);
    lcd.clear();

    auxilio(robo,id_aux);
    lcd.print("VELOCIDAD:");
    lcd.setCursor(0, 1);
    lcd.print(gps.f_speed_kmph());lcd.print(" Km/h");delay(2500);lcd.clear();

    tiempo++;delay(1000);

    if (tiempo >= 10){//////////////////////////// ACTUALIZA LA NUBE CADA 3 MINUTOS
      tiempo=0;
```

```
      auxilio(robo,id_aux);
      updateVariable(posicion,id_posicion,latitude,longitude,nombre,apellidos);
      auxilio(robo,id_aux);
      updateVariable(String(gps.f_speed_kmph()),id_velocidad,1.01,1.01,nombre,apellidos); //Actualizar variable VELOCIDAD
      auxilio(robo,id_aux);
    }
   // auxilio(robo,id_aux);}
    }}
}/////////////////////////////////////////////////////////----- ENVIO DATOS BOTONERAS

void auxilio(String variable,String id){

  if (digitalRead(inAccidente) == HIGH || digitalRead(inRobo) == HIGH){
      if (digitalRead(inAccidente) == HIGH){
          variable = "2";       digitalWrite(ledAux2, HIGH);   }
    else       digitalWrite(ledAux1, HIGH);
  lcd.clear();
  lcd.begin(16, 2);
  lcd.print("ENVIANDO SENAL");
  lcd.setCursor(0, 1);
  lcd.print("DE AUXILIO..");

  esp.println("AT+CGATT=1");
  delay(500);

  esp.println("AT+CSTT=\"internet.claro.com.ec\",\"\",\"\"");
  delay(500);
```

```
  esp.println("AT+CIICR");
  delay(1100);

  esp.println("AT+CIFSR");
  delay(1100);

  Serial.println("PREPARANDO ENVIO SEÑAL");
  String cmd = "AT+CIPSTART=\"TCP\",\"";
  cmd += IP; //IP del sitio a conectarse
  cmd += "\",8080";
  esp.println(cmd); //Crear conexion

  delay(2000); //Darle 2 segundos para responder

  cmd  = Post;
  cmd += id;//delay(80);
  cmd += id_f+"\r\n";//delay(80);
  cmd += token+"\r\n";
  cmd += Host+"\r\n";
  cmd += cc+"\r\n";
  cmd += ct+"\r\n"; //////longitud 1
  cmd += cl;
  cmd += "11\r\n\n";
  cmd += "{\"value\":";
  cmd += variable;
  cmd += "}\r\n";

```

```
  esp.print("AT+CIPSEND="); //Indicar cuantos datos se enviaran por TCP
  esp.println(cmd.length()); //Tamaño de los datos a enviar por TCP
  delay(2000);

  esp.print(cmd); //Enviar datos
  delay(11000);
  Serial.println("\n"+cmd+"\n");
  Serial.println("PETICION ENVIADA");

  esp.println("AT+CIPCLOSE");
  delay(850);
  esp.println("AT+CIPS");
  delay(850);

  lcd.clear();
  lcd.begin(16, 2);
  lcd.print("SENAL DE AUXILIO ENVIADA");
  lcd.setCursor(0, 1);
  lcd.print("    ENVIADA");
  delay(2500);lcd.clear();

  digitalWrite(ledAux1, LOW);
  digitalWrite(ledAux2, LOW);
  }
}////////////////////////////////////////////////////////-- VOID UPDATE VARIABLES

void updateVariable(String variable, String id, float latitude, float longitude, String nombre,String apellidos){
```

```
272
273  //  auxilio(robo,id_aux);
274  esp.println("AT");
275  delay(500);
276
277  esp.println("AT+CGATT=1");
278  delay(500);
279
280  esp.println("AT+CSTT=\"internet.claro.com.ec\",\"\",\"\"");
281  delay(500);
282
283  esp.println("AT+CIICR");
284  delay(2000);
285
286  esp.println("AT+CIFSR");
287  delay(2000);
288
289  esp.println("AT+CIPSTART=\"TCP\",\"things.ubidots.com\",\"8080\"");
290
291 delay(2000); //Darle 2 segundos para responder
292
293 Serial.println("CONEXION A SERVER");
294
295  int len2; String cmd;
296  if (variable == posicion ){
297    cmd = "{-value-:1,-context-:{-lat-:-2.1346860,-lng-:-79.9057390}}";
298    len2 = cmd.length();
```

```
299      }else if (variable == "187" ){
300        cmd = "{-value-:"+id_tarjeta+",-context-:{-nombre-:"+nombre+",-apellido-:"+apellidos+"}}";
301       len2 = cmd.length();
302        }else{
303          cmd = "xxxxxxxxxxxx";
304          len2 = variable.length() +cmd.length();
305             }
306
307     cmd  = Post;
308     cmd += id;//delay(80);
309     cmd += id_f+"\r\n";//delay(80);
310     cmd += token+"\r\n";
311     cmd += Host+"\r\n";
312     cmd += cc+"\r\n";
313     cmd += ct+"\r\n"; //////longitud 1
314     cmd += cl;
315     if (variable == posicion){
316     //cmd += "58\r\n\n";
317     cmd += String(len2)+"\r\n\n";
318     cmd += "{\"value\":1,";
319     cmd += "\"context\":";
320     cmd += "{\"lat\":";
321     cmd += String(latitude,7)+",";
322     cmd += "\"lng\":";
323     cmd += String(longitude,7)+"}";
324     cmd += "}\r\n";
325     }else if (variable == id_tarjeta){
```

```
      cmd += String(len2)+"\r\n\n";
      cmd += "{\"value\":";
      cmd += variable+",";
      cmd += "\"context\":";
      cmd += "{\"nombre\":";
      cmd += nombre+",";
      cmd += "\"apellido\":";
      cmd += apellidos+"}";
      cmd += "}\r\n";}
     else{
     cmd += String(len2)+"\r\n\n";
     cmd += "{\"value\":";
     cmd += variable;
     cmd += "}\r\n";
     }

    esp.print("AT+CIPSEND="); //Indicar cuantos datos se enviaran por TCP
    esp.println(cmd.length()); //Tamaño de los datos a enviar por TCP
    delay(1100);

    esp.println(cmd); //Enviar datos
    delay(1100);

    Serial.println(cmd);
    Serial.println("PETICION ENVIADA");

    esp.println("AT+CIPCLOSE");
```

```
    delay(800);
    esp.println("AT+CIPS");
    delay(800);
}

void loop(){
  Lector();
  delay(800);
  lcd.begin(16, 2);
  lcd.print("ESPERANDO A");
  lcd.setCursor(0, 1);
  lcd.print("USUARIO..");
  delay(2100);lcd.clear();
  Lector();
  //Serial.println(codigo);

  Serial.println("ESPERANDO USUARIO\n");
 if (codigo == id_tarjeta){
     digitalWrite(ledOk, HIGH);
     lcd.begin(16, 2);
     lcd.print("USUARIO");
     lcd.setCursor(0, 1);
     lcd.print("ACEPTADO..");
     Serial.println("USUARIO ACEPTADO");

     delay(2000);
     
```

```
     lcd.begin(16, 2);
     lcd.print("ENVIANDO DATOS ");
     lcd.setCursor(0, 1);
     lcd.print("DE USUARIO..");

    updateVariable(id_tarjeta,id_usuario,0.1,0.1,nombre,apellidos); //Actualizar variable
    lcd.clear();digitalWrite(ledOk, LOW);
  }
  while(codigo == id_tarjeta){

  GPS();}
}

```