ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES

TESIS PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

TEMA: DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE GESTIÓN AUTOMÁTICA PARA VEHÍCULOS SOBRE LA PLATAFORMA ANDROID

AUTOR: CASTILLO BAUTISTA, ANDRÉS FELIPE

DIRECTOR: ING. DARWIN ALULEMA

CODIRECTOR: ING. FREDDY ACOSTA

SANGOLQUÍ

2015

Certificado de Tutoría

UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE

INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

CERTIFICADO

Ing. Darwin Alulema

Ing. Freddy Acosta

CERTIFICAN

Que el trabajo titulado "DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE GESTIÓN AUTOMÁTICA PARA VEHÍCULOS SOBRE LA PLATAFORMA ANDROID", realizado por Andrés Felipe Castillo Bautista, ha sido guiado y revisado periódicamente y cumple las normas estatutarias establecidas por la Universidad de las Fuerzas Armadas – ESPE en su reglamento.

Debido a que se trata de un trabajo de investigación recomiendan su publicación.

Sangolquí, 24 de abril de 2015

Declaración de Responsabilidad

UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE

INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

DECLARACIÓN DE RESPONSABILIDAD

ANDRÉS FELIPE CASTILLO BAUTISTA

DECLARO QUE:

El proyecto de grado denominado "DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE GESTIÓN AUTOMÁTICA PARA VEHÍCULOS SOBRE LA PLATAFORMA ANDROID", ha sido desarrollado en base a una investigación exhaustiva, respetando derechos intelectuales de terceros, conforme las citas que constan al pie de las páginas correspondientes, cuyas fuentes se incorporan en la bibliografía.

Consecuentemente este trabajo es de mi autoría

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance científico del proyecto de grado en mención.

Sangolquí, 24 de abril de 2015

Andrés Felipe Castillo Bautista

Autorización de Publicación

UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE

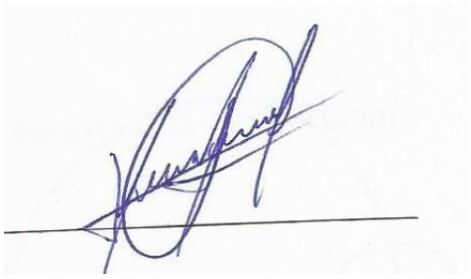
INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTORIZACIÓN

ANDRÉS FELIPE CASTILLO BAUTISTA

Autorizo a la Universidad de las Fuerzas Armadas – ESPE la publicación, en la biblioteca virtual de la Institución del trabajo "DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE GESTIÓN AUTOMÁTICA PARA VEHÍCULOS SOBRE LA PLATAFORMA ANDROID", cuyo contenido, ideas y criterios son de mi exclusiva responsabilidad y autoría.

Sangolquí, 24 de abril de 2015



Andrés Felipe Castillo Bautista

DEDICATORIA

El presente proyecto de grado se lo dedico a Dios, a mi familia y a todas aquellas personas que con su incondicional amor me han acompañado en este arduo camino académico sin abandonarme un solo instante. Este logro es tan mío como suyo.

Andrés F. Castillo Bautista

AGRADECIMIENTO

Agradezco a Dios por brindarme la fuerza y la sabiduría necesaria para enfrentar este desafiante trayecto.

A mi gran amor Nathaly quien recorrió este camino a mi lado, brindándome siempre su amor y comprensión aún en los momentos más difíciles.

A mi familia por el esfuerzo que han puesto en mi formación, el apoyo entregado y los valores inculcados en mí durante toda mi vida.

Finalmente agradezco al Ing. Darwin Alulema y al Ing. Freddy Acosta por su tiempo, su guía constante y todos los conocimientos compartidos que permitieron la realización del presente proyecto.

Andrés F. Castillo Bautista

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El presente proyecto describe el diseño y la implementación de un sistema encargado de realizar el monitoreo y diagnóstico automotriz de un vehículo mediante la utilización del estándar OBDII, una aplicación móvil sobre la plataforma Android y la implementación de un servidor Web desarrollado sobre lenguaje PHP. El sistema se encarga de extraer en tiempo real datos del funcionamiento del vehículo así como códigos de falla del mismo y los almacena remotamente en el servidor para que puedan ser analizados mediante un sitio Web. En el desarrollo se utilizó el terminal móvil Samsung Galaxy ACE, un vehículo Kia Rio R 2013 y el dispositivo de comunicación Bluetooth BAFX OBDII. En el diseño se determinaron los principales parámetros a monitorear en el vehículo, el soporte a los diversos protocolos de comunicación definidos en el estándar OBDII y las características y funcionalidades del sistema, además se diseñó e implementó una base de datos encargada de almacenar toda la información obtenida del vehículo. En base a los parámetros de diseño se desarrolló el software necesario para cumplir los requerimientos propuestos, además un simulador de códigos de falla fue implementado mediante una aplicación JAVA para permitir la realización de pruebas del sistema sin la necesidad de realizar una conexión real con el automóvil, los resultados de las pruebas realizadas al completar la implementación del sistema demostraron que el comportamiento y el desempeño alcanzado se enmarcan dentro del objetivos propuestos y el sistema es compatible con diversos modelos de autos y versiones de Android.

PALABRAS CLAVES:

- **OBDII**
- **ANDROID**
- **CÓDIGO DE FALLA**
- **MONITOREO**
- **PHP**

ABSTRACT

The present project describes the design and the implementation of an automotive diagnostic and monitoring system through OBDII standard, an Android application and a web server developed with PHP. The system read the car data in real time, read the car trouble codes and store them in the web server, all the information can be access with a web page. The used resources were a Samsung Galaxy Ace phone, a Kia Rio R 2013 as a test car and the BAFX ODII Bluetooth device. The design involves the setting of the main monitoring parameters and system features, also the support to the variety of communication protocols in OBDII standard was definied. A data base was designed and developed to store all the vehicle information. All the necessary software was developed based on the design parameters with the aim of reach the proposed requirements. A trouble code simulator developed in JAVA language also was implemented to test the applications without the actual connection between the car and the system, once the implementacion was complete, the test results show that the system performance reached aligns with the proposed objetives and the system is compatible with many cars and Android versions.

KEY WORDS:

- **OBDII**
- **ANDROID**
- **TROUBLE CODE**
- **MONITORING**
- **PHP**

CAPÍTULO 1

INTRODUCCIÓN

1.1 Antecedentes

Desde sus inicios hace más de 100 años el automóvil se ha posicionado como el medio de transporte predominante en nuestro mundo y se ha convertido en un instrumento clave en el funcionamiento de la sociedad. Sin embargo en las últimas décadas el mundo ha experimentado un intenso proceso de concientización ambiental, donde el automóvil tiene un rol significativo al ser una de las principales fuentes de dióxido de carbono que existen en el planeta.

Con el fin de monitorear y controlar la emisión de gases contaminantes producidos por vehículos de combustión interna y además permitir realizar un diagnóstico completo del estado mecánico del automotor, varios fabricantes desarrollaron sistemas de diagnóstico propios. Sin embargo, con el objetivo de asegurar la interoperabilidad entre sistemas se creó el estándar OBD (On Board Diagnostics). En su primera versión, OBDI se encargaba únicamente del monitoreo de los componentes esenciales del motor y su implementación en vehículos nuevos se volvió obligatoria en Estados Unidos desde 1991. [1] El estándar vigente OBDII se desarrolló en 1996 como resultado de medidas ambientales más estrictas en Estados Unidos y su implementación en automóviles nuevos ha sido obligatoria desde aquel año. [2] El estándar OBDII actualmente se encuentra implementado en la mayoría de vehículos nuevos a nivel global y es la principal herramienta para un diagnóstico automotriz completo. OBDII no es solo capaz de determinar errores en el funcionamiento de un vehículo, es capaz además de proporcionar información en tiempo real sobre distintos parámetros de los sistemas del mismo.

En sus inicios OBD funcionaba únicamente bajo comunicación serial mediante el estándar RS-232, actualmente incluso es posible acceder a la información del vehículo proporcionada por el estándar OBDII mediante tecnología Bluetooth.

En los últimos años los teléfonos inteligentes se han convertido en una poderosa plataforma para el desarrollo de aplicaciones de diverso tipo. La interacción con el usuario mediante pantallas táctiles, tamaño reducido y gran capacidad de procesamiento han convertido a los teléfonos inteligentes en computadoras de bolsillo. [3]

Desde su aparición en 2007, el sistema operativo Android ha incursionado en el mundo de los teléfonos inteligentes con gran éxito. Se estima que actualmente existen más de 1000 millones de dispositivos Android en el mundo. [4] Uno de los pilares del éxito de Android se basa en la gran cantidad de aplicaciones disponibles y en la relativa facilidad que implica el desarrollo de software para esta plataforma.

“Torque” es una de las aplicaciones más conocidas, dentro de las desarrolladas para Android que interactúan con el protocolo OBDII, la misma provee al usuario información en tiempo real de diversos parámetros del vehículo, así como también información sobre algunos errores en el funcionamiento del automotor, sin embargo “Torque” es una aplicación de uso personal. Únicamente el usuario dentro del automóvil es capaz de observar la información mostrada por el estándar OBDII y todos los datos recopilados no son almacenados para su posterior análisis.

1.2 Justificación e Importancia

De acuerdo al último informe sobre la situación mundial de la seguridad vial, publicado por la Organización Mundial de la Salud; Ecuador ocupa el segundo lugar en América Latina en mayor tasa de muertes por accidentes de tránsito con una media de 28 muertes por cada 100000 habitantes. [5]

Según la Agencia Nacional de Tránsito la imprudencia e impericia es la principal causa de accidentes en nuestro país, sin embargo fallas mecánicas y el mal estado de los automotores constituyen también una importante causa en el desarrollo de accidentes. [6]

Actualmente muchas empresas requieren que sus vehículos estén en constante utilización, esto provoca que estén expuestos a sufrir fallos mecánicos capaces de comprometer no solo la integridad de la unidad sino además la seguridad de los ocupantes. Estos fallos además pueden provocar la paralización del vehículo por reparación lo que representa un problema en cuanto a costos y logística para los propietarios. Un sistema que integre la información provista por el estándar OBDII con una interfaz de usuario amigable, permitiría a empresas y a propietarios de vehículos en general estar al tanto del estado mecánico de la unidad, siendo capaces de identificar y resolver problemas presentes en los automotores y prevenir potenciales daños catastróficos en los mismos.

La implementación de este sistema permite asegurar la confiabilidad de los vehículos y garantizar la seguridad de los ocupantes en lo que respecta a la parte mecánica.

1.3 Alcance del proyecto

El proyecto pretende entregar como producto final un sistema que permita el monitoreo del estado mecánico de una flota de vehículos a partir de un dispositivo OBDII de marca BAFX [7] conectado mediante tecnología Bluetooth a un Smarthphone con sistema operativo Android y este a su vez conectado a un servidor mediante Internet.

El dispositivo OBDII se conecta físicamente al vehículo mediante el puerto OBDII propio del automotor.

Se desarrolla una aplicación móvil sobre la plataforma Android encargada de:

- Interpretar y procesar la información del vehículo provista por el dispositivo OBDII.
- Mostrar la información al conductor o usuario del vehículo utilizando la pantalla del teléfono.
- Transmitir dicha información a un servidor remoto mediante Internet.

Se implementa además un servidor WEB, en el cual se diseñó y creó una base de datos encargada de almacenar la información provista por el teléfono inteligente sobre el estado mecánico del automotor. Se implementa además una aplicación WEB mediante la cual se podrá realizar el monitoreo remoto de la información desde cualquier computador.

En la siguiente figura se muestra el esquema del sistema implementado.

Figura. 1.1 Esquema del sistema de monitoreo automotriz

1.4 Objetivos

1.4.1 Objetivo General

- Implementar un sistema de gestión automática para vehículos que permita monitorear el estado mecánico de un automotor tanto de forma local como remota utilizando un dispositivo OBDII y un teléfono inteligente Android.

1.4.2 Objetivos Específicos

- Realizar un estudio acerca del protocolo OBDII; sus características, modos de operación y métodos de identificación de fallas en automotores.
- Estudiar el estándar WPAN 802.15.1 (Bluetooth).
- Analizar las propiedades y capacidades del sistema operativo Android.
- Desarrollar una aplicación para dispositivos móviles en la cual los ocupantes visualicen el estado mecánico del vehículo.
- Desarrollar una aplicación en el servidor para el monitoreo remoto del estado mecánico del vehículo.
- Evaluar el desempeño del sistema de monitoreo.

CAPÍTULO 2

MARCO TEÓRICO

2.1 Estándar OBDII

En el año de 1988, la Agencia de Protección Ambiental Estadounidense, EPA por sus siglas en inglés, estableció a los fabricantes de vehículos incluir obligatoriamente un programa de auto-diagnóstico en las computadoras de los automotores. Esta primera generación de sistemas de diagnóstico a bordo se conoció como OBDI.

OBDI constituye un conjunto de instrucciones programadas en las computadoras o cerebros de los automotores. El objetivo principal de estos programas era detectar algún desperfecto que pudiese producirse en actuadores, switches o cableado de cualquier sistema que tenga relación con las emisiones de gases por parte del vehículo. Si la computadora detectaba un fallo o problema de funcionamiento en cualquiera de estos sistemas, un indicador en el tablero de instrumentos se encendía. El indicador por tanto solo se encendía cuando se detectaba un problema en las emisiones del vehículo.

OBDII representa la segunda versión de OBDI, los programas de diagnóstico son mejorados y presentan una nueva función de monitoreo del automotor a diferencia de OBDI capaz únicamente de identificar componentes dañados. Esta función de monitoreo no solo se ocupa de sistemas relacionados con emisiones sino de varios otros responsables del correcto funcionamiento y seguridad del vehículo. OBDII además permite que toda esta información esté disponible y sea accesible mediante el equipo adecuado a mecánicos y propietarios. Sin embargo la característica principal de OBDII es la estandarización, OBDI fue implementado por cada fabricante en base a sus propias consideraciones. El conector, los protocolos de comunicación, códigos

de fallas y terminología variaban según la marca del vehículos por lo que los sistemas de diagnóstico no tenían interoperabilidad entre automóviles de distintos fabricantes. Por tanto los tres objetivos principales de OBDII son:

- Estandarizar los procedimientos de comunicación y protocolos entre los equipos de diagnóstico y las computadoras de los automotores.
- Promover el uso de un conector de enlace estándar en todos los vehículos.
- Estandarizar los números de código, definiciones de código y lenguaje usado para la descripción e identificación de fallas en el automóvil.

Actualmente la mayoría de vehículos livianos modernos incorporan el estándar OBDII, sin embargo para vehículos pesados su implementación aún no es obligatoria.

2.1.1 Conector OBDII

El conector de enlace de datos o DLC por sus siglas en inglés constituye la interfaz física entre la computadora del vehículo y el sistema o equipo de diagnóstico. En los sistemas OBDI la forma, tamaño y ubicación del conector variaba de fabricante a fabricante. Con OBDII el conector de 16 pines está estandarizado y si bien su ubicación varía entre vehículos, generalmente es posible encontrarlo en la parte inferior izquierda del tablero de instrumentos.

La mayoría de los pines se encuentran asociados a características o funciones específicas del estándar OBDII y deben ser implementadas de forma obligatoria en todos los vehículos, estos pines proveen de la alimentación necesaria a los equipos de diagnóstico mediante la batería del vehículo así como los buses de datos para los protocolos de comunicación OBDII. Existen además varios pines sin asignación definida en el estándar, dichos pines son definidos por el fabricante y tienen como propósito brindar acceso a información complementaria sobre diversos sistemas del automóvil que tengan relevancia de acuerdo a las consideraciones de cada fabricante.

La distribución de pines en el DLC OBDII se muestra en la figura 2.1, mientras la asignación de los mismos se presenta en la tabla 2.1.

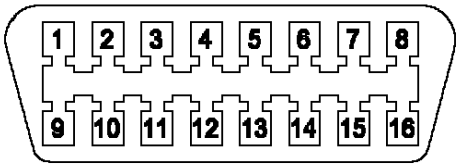


Figura. 2.1 Distribución de pines en conector OBDII. [8]

Tabla. 2.1 Asignación de pines conector OBDII

Pin	Función	Pin	Función
1	Sin asignar	9	Sin asignar
2	J1850 Bus+	10	J1850 Bus-
3	Sin asignar	11	Sin asignar
4	Tierra (chasis)	12	Sin asignar
5	Tierra (señal)	13	Sin asignar
6	CAN High (J-2284)	14	CAN Low (J-2284)
7	ISO 9141-2 (K Line)	15	ISO 9141-2 (L Line)
8	Sin asignar	16	Alimentación

2.1.2 Protocolos de comunicación

El protocolo de comunicación representa como los datos son intercambiados entre la computadora a bordo del vehículo y el dispositivo de diagnóstico exterior.

La sociedad de ingenieros automotrices, SAE por sus siglas en inglés, ha definido tres clasificaciones para los protocolos de comunicación basándose en la velocidad a la cual los datos del automotor pueden ser transferidos:

- Clase A: Velocidad de transmisión de hasta 10 Kb/s.
- Clase B: Velocidad de transmisión de hasta 100 Kb/s.
- Clase C. Velocidad de transmisión de hasta 1 Mb/s.

Actualmente existen cinco protocolos de comunicación definidos para ser utilizados en el estándar OBDII. Un vehículo puede utilizar cualquiera de estos cinco protocolos:

- ISO 9141-2
- SAE J1850 (VPW 10.4K)
- SAE J1850 (PWM 41.6K)
- ISO 14230-4 KW 2000
- ISO 15765 CAN (Control Area Network)

2.1.3 Modos de operación

El estándar OBDII posee diez modos de operación definidos de acuerdo a qué tipo de información el mecánico o conductor requiere. Cada uno de estos modos se identifica por un valor hexadecimal de 01 a 0A. Los modos de operación se muestran en la tabla 2.2.

Tabla. 2.2 Modos de operación de OBDII

Modo	Descripción
01	Monitoreo de datos instantáneos
02	Monitoreo de datos congelados
03	Mostrar códigos de falla diagnosticados almacenados
04	Borrar códigos de falla diagnosticados
05	Monitoreo de sensor de oxígeno
06	Monitoreo de otros sistemas
07	Mostrar códigos de falla diagnosticados pendientes
08	Control de la operación del sistema de diagnóstico a bordo
09	Solicitar información de identificación del vehículo
0A	Mostrar códigos de falla diagnosticados permanentes

Cada fabricante puede agregar modos según considere, con el fin de proporcionar acceso a datos del vehículo relacionados a tecnología propia de la marca o del automotor.

De estos diez modos, dos destacan por la relevancia de la información que proveen. El modo 01 permite mediante un identificador de parámetro, PID por sus siglas en inglés, obtener datos en tiempo real sobre el funcionamiento del automotor, mientras el modo 03 permite recopilar los códigos de falla que presenta el vehículo, estos códigos a su vez pueden ser interpretados para entender el origen o motivo de la falla que se muestra.

2.1.4 Códigos de falla OBDII

Llamados también códigos de problemas diagnosticados o DTC por sus siglas en inglés. Son códigos que identifican fallas o funcionamiento defectuoso en sistemas o componentes específicos de un vehículo.

Cada vehículo posee en su tablero de instrumentos un indicador de mal funcionamiento también conocido como “Engine Light”. Cuando la computadora del automotor detecta un problema en el funcionamiento de uno o más sistemas: Asigna un código de falla que permite identificar el origen de dicho problema, almacena este código en la memoria interna de la computadora y enciende el indicador de mal funcionamiento con el fin de informar al propietario que debe realizar un chequeo del vehículo. En la figura 2.2 se muestran varios ejemplos de indicadores de mal funcionamiento.

Figura. 2.2 Indicadores de mal funcionamiento. [9]

Los códigos de falla implementados en OBDII se clasifican en dos tipos:

- Códigos genéricos.
- Códigos específicos del fabricante.

Los códigos genéricos, son aquellos que son usados por todos los fabricantes, las definiciones de los mismos son las mismas sin importar la marca o modelo del vehículo y son establecidos por la SAE en conjunto con el organismo internacional de estandarización ISO.

Los códigos específicos del fabricante por su parte son códigos cuyo control y definición está dado por el fabricante, el estándar OBDII no requiere la implementación de códigos adicionales a los genéricos, sin embargo muchos fabricantes los implementan con el objetivo de facilitar el diagnóstico en los sistemas de sus automóviles.

Los códigos de fallas están compuestos de cinco caracteres alfanuméricos, en la figura 2.3 se muestra la estructura de un código de falla estándar.

Figura. 2.3 Estructura de un código de falla

El primer caracter es una letra, la misma que identifica el sistema principal donde ocurrió el problema:

- B: Carrocería.
- C: Chasis.
- P: Tren motriz.
- U: Red.

El segundo caracter es un número, se encarga de identificar el tipo de código de falla:

- 0: Códigos genéricos.
- 1: Códigos específicos del fabricante.
- 2: Incluye códigos tanto genéricos como del fabricante.

- 3: Incluye códigos tanto genéricos como del fabricante.

El tercer caracter de igual forma es un número mediante el cual se identifica de forma específica el sistema o subsistema donde se produjo el problema:

- 1: Medición de aire y combustible.
- 2: Medición de aire y combustible (Únicamente identifica mal funcionamiento del circuito del inyector).
- 3: Sistema de ignición.
- 4: Sistema de control de emisiones auxiliar.
- 5: Control de velocidad del vehículo y sistema de control en modo inactivo.
- 6: Circuitos de salidas del computador.
- 7: Transmisión.
- 8: Transmisión.

El cuarto y quinto caracter son también dígitos numéricos, estos establecen la sección del sistema donde se produjo la falla.

Por ejemplo si la computadora del vehículo devuelve el código P0201, representa que existe un problema en el circuito del inyector del cilindro 1. Se puede llegar a esta interpretación gracias a los cinco caracteres del código de falla como se muestra en la figura 2.4.

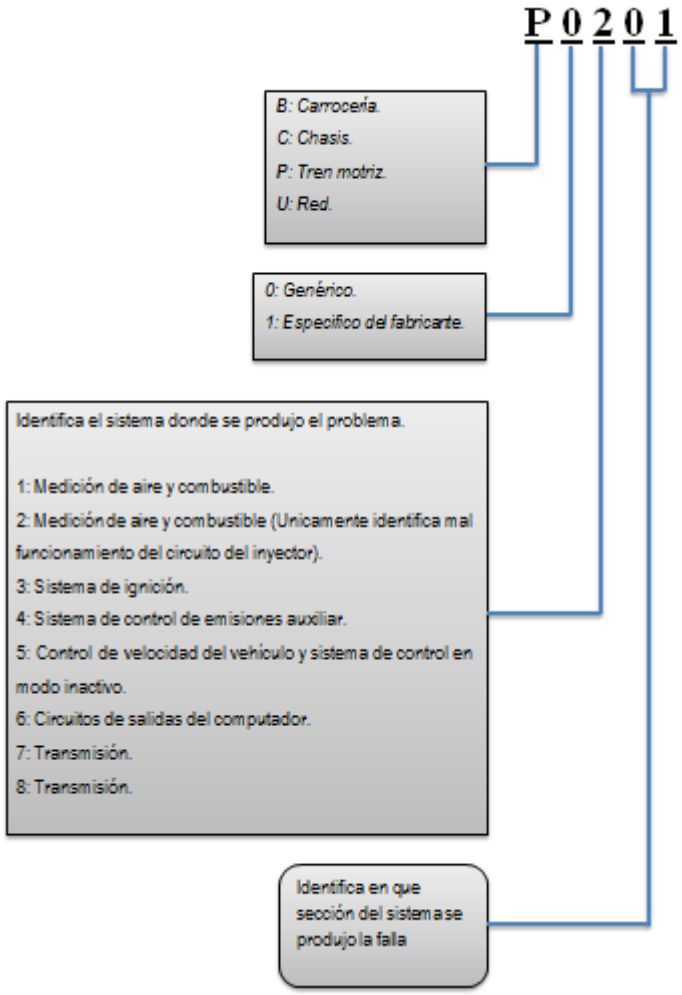


Figura. 2.4 Interpretación de códigos de falla

Si bien todos los vehículos que tienen implementado el estándar OBDII manejan el formato presentado, el único grupo de códigos obligatorio es el grupo de tren motriz. En Ecuador la mayoría de vehículos comercializados únicamente arrojan códigos de fallas correspondientes a dicho grupo.

2.2 Descripción del dispositivo BAFX OBDII

El adaptador bluetooth BAFX OBDII es un dispositivo que permite la sincronización y comunicación entre la computadora de cualquier vehículo fabricado

a partir de 1996 y un dispositivo Android o un computador con sistema operativo Windows. En la figura 2.5 se muestra el dispositivo BAFX OBDII.

Figura. 2.5 Adaptador BAFX OBII

El adaptador BAFX OBDII soporta todos los protocolos de comunicación definidos en el estándar OBDII:

- ISO 9141-2
- SAE J1850 (VPW 10.4K)
- SAE J1850 (PWM 41.6K)
- ISO 14230-4 KW 2000
- ISO 15765 CAN (Control Area Network)

Este dispositivo constituye únicamente la interfaz física de comunicación entre la computadora del vehículo y un dispositivo externo. Para poder recopilar información del vehículo y obtener los códigos de falla es necesaria la implementación de software dedicado a la interpretación de las tramas enviadas por el dispositivo.

2.3 Estándar 802.15.1

A inicios del siglo XXI se tenía la visión de una tecnología basada en radio de bajo costo y bajo consumo que sería capaz de conectar dispositivos sin la necesidad de un cable, formando redes de área personal similares a las redes de área local cableadas. El consorcio Bluetooth es el resultado de la unión de varios fabricantes con el fin de crear una especificación abierta que vuelva realidad esta visión. [10]

802.15.1 es una especificación orientada a la industria que define la conectividad basada en radio frecuencia de corto alcance para dispositivos personales portátiles. La especificación 802.15.1 es el referente para todo fabricante que desee implementar dispositivos Bluetooth, en la misma se definen las capas más bajas de la tecnología inalámbrica Bluetooth. [11]

La tecnología de transmisión en la cual se basa Bluetooth es espectro ensanchado por salto de frecuencia o FHSS por sus siglas en inglés. FHSS se puede definir como una técnica de modulación en la cual la señal es transmitida saltando de frecuencia en frecuencia de forma pseudoaleatoria. Tanto el transmisor como el receptor conocen el orden en el que se van a producir los saltos en la comunicación. [12]

En lo referente a la arquitectura, Bluetooth tiene su base en el concepto de picored. Picored puede definirse como una red inalámbrica de corto alcance compuesta de al menos dos equipos que se comunican de forma sincrónica usando el mismo salto de frecuencia.

El hardware necesario para el establecimiento de comunicación Bluetooth consiste principalmente de un transceiver de radio frecuencia, una antena, un módulo de control del enlace y software de control y gestión. Cuando un equipo cumple con todas las especificaciones de hardware y software Bluetooth, puede implementarse y venderse en el mercado garantizándose la interoperabilidad de dispositivos.

2.4 Plataforma Android

Android es un conjunto de software de código abierto destinado a terminales móviles, es independiente del factor de forma de dichos terminales y actualmente es el sistema operativo más popular a nivel global. [13]

Un pilar fundamental para su éxito radica en la característica de ser una plataforma de software libre, esta cualidad ha permitido a usuarios y desarrolladores potenciar y maximizar el aprovechamiento de las capacidades del terminal. Además muchas de las aplicaciones desarrolladas para esta plataforma han revolucionado el mundo de las comunicaciones en los últimos seis años. Android se caracteriza

además por ser software multiplataforma, puede instalarse en terminales inteligentes independientemente del fabricante, tamaño de pantalla u operador de servicios.

2.4.1 Bluetooth API

Dentro de la plataforma de desarrollo para Android se incluye una librería específica para la implementación de la transmisión Bluetooth, dicha librería se denomina Bluetooth API.

Bluetooth API permite desarrollar aplicaciones en las cuales un dispositivo es capaz de intercambiar información de manera inalámbrica utilizando tecnología Bluetooth. Además es posible mediante esta librería implementar comunicación de tipo punto-punto o punto-multipunto. [14] Las principales acciones que se pueden realizar con el uso de Bluetooth API en una aplicación son:

- Buscar dispositivos Bluetooth cercanos.
- Acceder a la lista de dispositivos Bluetooth anclados.
- Establecer canales de comunicación entre dispositivos.
- Enviar y recibir datos hacia y desde otros dispositivos.
- Gestionar conexiones múltiples.

El Bluetooth API posee clases e interfaces propias que facilitan el manejo de comunicaciones con tecnología 802.15.1.

Bluetooth Adapter

Esta clase representa al adaptador Bluetooth local del teléfono. Es el punto de partida para el establecimiento y gestión de las conexiones ya que es la clase encargada de buscar nuevos dispositivos, consultar los dispositivos emparejados e instanciar objetos de la clase BluetoothDevice.

BluetoothDevice

Es la clase que representa a un dispositivo Bluetooth remoto. Se utiliza un objeto de esta clase para solicitar la conexión con un dispositivo específico mediante un objeto de la clase BluetoothSocket. Además la clase BluetoothDevice permite

realizar consultas sobre los dispositivos remotos, estas consultas pueden ser el nombre, la dirección MAC o el estado del dispositivo.

BluetoothSocket

Un objeto de esta clase representa al punto de conexión que permite el intercambio de información entre dos dispositivos Bluetooth.

BluetoothServerSocket

Un objeto BluetoothServerSocket permite la conexión entre dos dispositivos Android. Uno de los dispositivos debe abrir un ServerSocket con esta clase cuando otro terminal realiza una solicitud de conexión. Si la conexión es aceptada se devuelve un objeto de la clase.

2.5 Expresiones Regulares

Una expresión regular es un conjunto de caracteres que representan a un grupo de cadenas de texto bajo ciertos parámetros y criterios comunes. Es decir permiten la búsqueda de texto en función de patrones definidos por la expresión regular. [15] Se las conoce también como RegEx y hacen uso de un lenguaje específico para lograr la generación de patrones. Mediante el uso de expresiones regulares es posible extraer partes específicas de un texto, es posible encontrar palabras dentro de una frase o incluso un documento.

En programación el uso de expresiones regulares permite localizar cadenas específicas dentro de millones de caracteres. Sin embargo para tener soporte de expresiones regulares es necesario contar con un motor de búsqueda especializado. Los lenguajes de programación que ofrecen soporte para expresiones regulares son:

- Java
- JavaScript
- Perl
- Php
- Python
- Net.Framework

2.5.1 Descripción de caracteres especiales en RegEx

Caracter punto “ . ”

Si dentro de una expresión regular se encuentra el carácter “.”, el motor de búsqueda lo entenderá como cualquier caracter presente una sola vez. Por ejemplo si la expresión regular es “p.r” y se aplica una búsqueda sobre la frase “El perro espera la llegada de su dueño por varias horas”. El motor de búsqueda arrojará como coincidencia las cadenas “per”, “per” y “por”. El punto ubicado entre el caracter “p” y el caracter “r” simboliza que toda cadena que contenga cualquier carácter en medio de estas dos letras arrojará una coincidencia de búsqueda.

Carácter Barra inversa “ \ ”

El caracter barra inversa por sí solo no tiene significado. Es necesario usarlo junto a otros caracteres para que tenga funcionalidad. Si se coloca una barra inversa antes de un caracter especial como el punto, este pierde su significado como caracter especial y pasa a ser simplemente el caracter punto. Análogamente si se coloca una barra inversa antes de ciertos caracteres normales, estos adquieren una funcionalidad especial. Dichos caracteres y el significado especial que adquieren son:

- \t Tabulación horizontal
- \r Retorno de carro
- \n Salto de línea
- \a Representa una notificación de audio
- \e Tecla escape
- \f Salto de página
- \v Tabulación vertical
- \d Cualquier dígito del 0 al 9
- \D Cualquier carácter excepto dígitos del 0 al 9
- \w Cualquier carácter alfanumérico
- \W Cualquier carácter que no sea alfanumérico
- \s Espacio en blanco
- \S Cualquier carácter excepto espacio en blanco

Corchetes “[ ]”

Los corchetes permiten la agrupación de caracteres con el fin de encontrar coincidencia entre algunas opciones posibles. Dentro de los corchetes los caracteres especiales excepto los caracteres con barra inversa pierden su significado y se comportan como caracteres literales. Por ejemplo el carácter punto dentro de corchetes representará de forma literal un punto. Por ejemplo si se realiza una búsqueda con la expresión “[aá]rbol” las coincidencias arrojadas en la búsqueda serán las palabras “árbol” o “árbol”. Los paréntesis indican que el primer caracter de la palabra puede ser “a” o “á” pero no ambas.

También es posible establecer rangos de caracteres mediante un guión. Por ejemplo si se requiere expresar un número hexadecimal, la expresión regular usada sería “[A-Fa-f0-9]”. Dentro del paréntesis existen tres rangos posibles, por tanto al realizar una búsqueda los resultados que se arrojarán como coincidencia serán los caracteres que representan cantidades hexadecimales de 0 a F.

Paréntesis “( )”

Los paréntesis funcionan de manera similar a los corchetes con la diferencia que los caracteres especiales dentro de los paréntesis conservan su significado especial contrario a lo que sucede en los corchetes

Llaves “{ }”

Las llaves por sí solas no presentan ninguna funcionalidad, es necesario de dos condiciones para apreciar la utilidad de estos caracteres. Dentro de las llaves debe existir un número o rango de números separados por comas y las llaves deben ubicarse a la derecha de otra expresión. Los números dentro de los corchetes indican el número de repeticiones de la expresión ubicada a la izquierda de los corchetes. Por ejemplo si se tiene la expresión regular “[a-zA-Z]{3,5}” los resultados de esta expresión serán todas aquellas cadenas de caracteres que contengan letras mayúsculas o minúsculas con una longitud mínima de 3 caracteres y una máxima de 5 caracteres. Las cadenas “Hola”, “día”, “FUEGO” son ejemplos de resultados que arrojaría esta expresión.

Caracter “$”

El carácter “$” representa el final de una línea. Por ejemplo si se necesita diferenciar entre “punto seguido” y “punto aparte” es posible utilizar la expresión regular “\.$”. La barra inversa colocada antes del punto indica que se tome al punto no como caracter especial sino solo como el caracter punto y el símbolo “$” indica que solo se busque aquellos puntos que se encuentren en el final de una línea.

Caracter “^”

Este caracter posee dos significados dependiendo de la forma en que se lo utilice en la expresión regular. Si se lo utiliza por sí solo representa el inicio de una línea de forma similar al signo “$” que representa el final de una línea. Por ejemplo si se toma la expresión “^[A-Z]” lo que se está buscando son todas las líneas que comiencen con una letra mayúscula.

La otra funcionalidad de este caracter se presenta cuando está ubicado dentro de corchetes y al lado izquierdo de otra expresión. Cuando se presenta este escenario el caracter indica que debe excluirse del resultado a las cadenas que coincidan con la expresión ubicada a la derecha de “^”. Por ejemplo la expresión “[^A-Z]” buscará cualquier carácter distinto de cualquier letra mayúscula.

Signo de interrogación “?”

El caracter “?” indica que parte de la expresión regular puede o no estar presente, es decir es una porción opcional de la búsqueda. Por ejemplo si se tiene la expresión “H?ola”, los resultados de la búsqueda serán las palabras Hola y ola.

Caracter “*”

El asterisco permite encontrar repeticiones de una cadena que puede o no estar presente. Es decir cadenas de texto que están repetidas cero o más veces. Por ejemplo la expresión “buenos\s*días” arrojará como resultados todas aquellas cadenas que cuenten con cero, uno o varios espacios en blanco entre las dos palabras como “buenosdías”, “buenos días” o “buenos días”.

Caracter “+”

Similar al caracter “*” con la diferencia que la cadena de texto puede estar repetida una o más veces a diferencia del asterisco en la que puede estar repetida cero o más veces. La expresión “buenos\s+días” por ejemplo devolverá como resultado toda cadena que contenga al menos un espacio en blanco entre las dos palabras.

Caracter “|”

Este caracter permite realizar una búsqueda entre varias opciones. Por ejemplo la expresión “y|o” permitirá encontrar los caracteres “y” u “o”.

2.6 Base de Datos

Una base de datos puede definirse como un sistema computarizado de almacenamiento y organización de registros que permite a los usuarios modificar y recuperar la información almacenada en dichos registros. [16]

Las operaciones principales que un usuario puede realizar sobre una base de datos son:

- Añadir nuevos archivos a la base de datos
- Eliminar archivos existentes en la base de datos
- Insertar datos en los archivos existentes
- Eliminar datos en los archivos existentes
- Recuperar datos
- Modificar datos

2.6.1 Conceptos básicos

Entidad

Se conoce como entidad a cualquier objeto representable y distinguible dentro de la base de datos.

Relación

Una relación es el medio de vinculación de dos entidades, es bidireccional, es decir relaciona las entidades tanto en un sentido como el otro. Además una relación representa un objeto dentro de la base de datos, por lo que encaja perfectamente también dentro del concepto de entidad.

Atributos

Son las características propias de cada entidad que la definen como tal. Cada entidad tiene uno o varios atributos que la representan. Por ejemplo en la entidad auto se pueden establecer varios atributos que permiten definirlo como tal, estos atributos podrían ser la placa del vehículo, la marca, el modelo o el color.

Cardinalidad

En una relación, la cardinalidad determina el número de entidades con las cuales tiene relación una entidad específica. La cardinalidad puede tomar cuatro formatos posibles:

- Uno a Uno: Una entidad A se relaciona únicamente con una entidad B.
- Uno a Varios: Una entidad A se relaciona con ninguna o varias entidades B.
- Varios a Uno: Ninguna o varias entidades A se relacionan solo con una entidad B.
- Varios a Varios: Ninguna o varias entidades A se relaciona con ninguna o varias entidades B.

Sistema de administración de base de datos (DBMS)

Es el componente de software principal en una base de datos, es el encargado de gestionar las solicitudes de acceso a la base de datos, así como también procesar las operaciones de inserción, eliminación y modificación de datos. Es además la interfaz entre los usuarios del sistema y la base de datos a nivel físico o de hardware. [16]

Modelo de datos relacional

El modelo de datos relacional es un modelo en el cual la interpretación entre los datos y la base de datos es sumamente directa. Mediante una concepción lógica del sistema es posible estructurar la base de datos requerida.

En un modelo relacional las entidades son representadas como tablas dentro de las cuales se encuentra la información de la base de datos. Cuando una operación se realiza sobre una tabla, como por ejemplo la recuperación de ciertos datos, una nueva tabla es generada para mostrar únicamente los datos solicitados.

2.6.2 Modelo conceptual de una base de datos (CDM)

Es el diagrama inicial que se realiza en el diseño de una base de datos. No muestra información a detalle de cómo se conformará toda la base pero si determina la estructura básica de la misma. El CDM permite al desarrollador establecer la línea base sobre la cual se realizará el diseño, es crucialmente importante que el modelo conceptual se estructure correctamente ya que de lo contrario, la base de datos podría ocasionar problemas a los usuarios a futuro. [17]

En el modelo conceptual se incluyen únicamente las entidades presentes en la base de datos, la relación entre entidades y la cardinalidad. [18]

2.6.3 Modelo físico de una base de datos (PDM)

El modelo físico es un diagrama que describe como serán almacenada y organizada la información en una base de datos. Toma como partida el modelo conceptual, sin embargo en el modelo físico se especifica todas las tablas y atributos que deberán estar presentes en la base de datos.

El modelo físico muestra al desarrollador la estructura real de la base de datos con un gran nivel de detalle sobre cómo se realiza la relación entre las entidades.

2.6.4 MySQL

MySQL es un sistema gestor de bases de datos de código abierto, es uno de los más populares en el mundo y se estima que se han distribuido 100 millones de copias del software a lo largo de la historia. [19]

Algunas de las principales ventajas de MySQL son:

- Portabilidad
- Disponibilidad de código fuente
- Fácil aprendizaje
- Costo reducido
- Alto desempeño

SQL

MySQL está basado en SQL. SQL es un lenguaje de acceso a base de datos que permite realizar consultas, inserción, modificación o eliminación de registros. SQL es un lenguaje que se basa en declaraciones, es decir expone lo que requiere obtener, mas no como obtenerlo.

Las principales sentencias utilizadas en el lenguaje SQL para el manejo de una base de datos son:

- SELECT: Permite la consulta de registros almacenados en la base de datos.
- INSERT: Posibilita insertar nuevos registros en las tablas de una base de datos.
- UPDATE: Permite modificar registros existentes en la base de datos.
- DELETE: Elimina registros de una base de datos.

2.7 Lenguaje PHP

PHP es un lenguaje de programación de tipo Script orientado a servidores y diseñado principalmente para el desarrollo de aplicaciones web. Es posible incluir porciones de código PHP dentro de código HTML, cada vez que la página sea visitada, el código PHP incluido en la página web será ejecutado. [20]

PHP es un producto de código abierto, es posible tener acceso al código fuente, modificarlo y distribuirlo libremente.

Los principales competidores del lenguaje PHP son:

- JSP
- ASP
- PERL
- Allaire ColdFusion

En comparación con los lenguajes mencionados, PHP presente algunas ventajas:

- Portabilidad
- Costo reducido
- Interfaces con la mayoría de sistemas gestores de base de datos
- Uso y aprendizaje relativamente sencillo
- Librerías ya implementadas para tareas simples y comunes

PHP es un lenguaje eficiente, usando únicamente un servidor web es posible procesar una gran cantidad de visitas por día. Además dado que fue diseñado para la creación de sitios web, posee varias herramientas útiles incorporadas. Ejemplos de estas herramientas son: generar imágenes GIF, conectar con otros servicios de red, generar documentos e incluso enviar emails.

CAPÍTULO 3

DISEÑO Y DESARROLLO

3.1 Estructura del sistema

El sistema de gestión para vehículos sobre la plataforma Android toma como punto de partida el protocolo OBDII, mediante dicho protocolo se obtienen los datos necesarios del automotor que son procesados, transmitidos y presentados a los usuarios.

El sistema posee dos modos de operación, el primero como un monitor local del funcionamiento del vehículo y el segundo como un escáner de fallas cuyos resultados pueden ser visualizados de forma remota. En el modo de monitoreo local se engloba la lectura y procesamiento de la información provista por la computadora del vehículo mediante el dispositivo Bluetooth y la presentación de datos al conductor a través de la pantalla del teléfono inteligente. En el modo de escáner el acceso a la información de forma remota está compuesto por un desarrollo Web que interactúa con el teléfono y el usuario, obteniendo y presentando datos provenientes de la computadora del vehículo.

Se utiliza un modelo de tipo Cliente-Servidor para el desarrollo del sistema, bajo este modelo varios dispositivos son capaces de enviar datos a un único servidor, el mismo que gestiona y organiza la información para su presentación final [21]. En la figura 3.1 se muestra la estructura del sistema bajo el modelo cliente servidor.

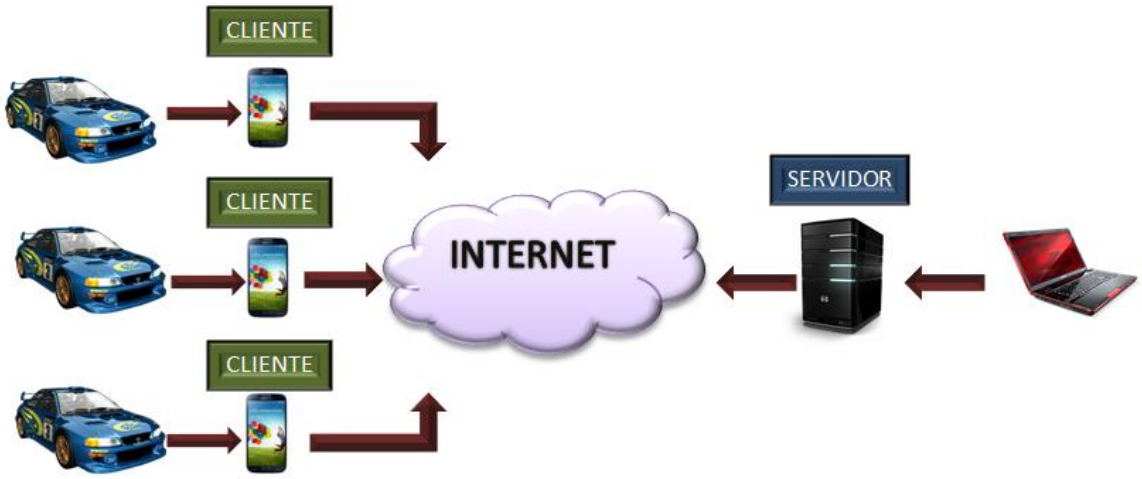


Figura. 3.1 Estructura del sistema.

El sistema está conformado por los siguientes componentes:

- Aplicación móvil para dispositivos Android, versión 2.3 en adelante.
- Aplicación Web desarrollada en lenguaje PHP.
- Base de datos MySQL.

3.1.1 Requerimientos

La aplicación debe cumplir con tres objetivos principales, obtener y presentar parámetros del funcionamiento del vehículo al conductor, realizar un diagnóstico en busca de fallas en el automóvil y enviar esta información a un servidor donde será almacenada en una base de datos.

Dentro del conjunto de parámetros que pueden ser obtenidos mediante OBDII, se eligieron específicamente cinco dada su importancia y utilidad al momento de conducir un vehículo. Los valores de estos parámetros son presentados al conductor mediante la pantalla del teléfono y enviados al servidor web cada minuto. La tabla 3.1 muestra la definición de cada elemento medido en la aplicación.

Tabla. 3.1 Parámetros de monitoreo local

Parámetro	Unidades	Descripción
Revoluciones por minuto	RPM	Velocidad de giro del motor, este valor es indispensable conocer para no llevar a límites peligrosos al motor.
Velocidad	Km/h	Velocidad del automóvil.
Carga Motor	%	Porcentaje que representa el trabajo que está realizando el motor al desplazarse
Temperatura Refrigerante	°C	Indica la temperatura a la cual está trabajando el motor. Valores elevados de este parámetro pueden provocar daños irreversibles en el vehículo.
Posición Acelerador	%	Porcentaje de apertura del sistema de admisión. Influye directamente en el consumo de combustible.

Con estos cinco parámetros definidos y con el fin de lograr una correcta gestión y organización de la información que es recopilada, varios requerimientos deben ser cumplidos por los diferentes componentes del sistema.

Requerimientos en la aplicación móvil

Los requerimientos que debe cumplir la aplicación móvil son:

- Registro de automóviles: El usuario debe especificar la placa, marca y modelo del vehículo.
- Búsqueda y emparejamiento Bluetooth: El teléfono debe ser capaz de encontrar nuevos dispositivos Bluetooth y establecer comunicación con los mismos.
- Interpretación de tramas provenientes del dispositivo Bluetooth: La aplicación tiene la capacidad de comunicarse con el vehículo e interpretar las tramas provenientes del mismo, procesar dichas tramas y presentar la información.
- Comunicación con el servidor: Cuando se requiera, la aplicación debe enviar a través de internet hacia el servidor la información extraída del vehículo.

Requerimientos en la aplicación Web

La aplicación Web posee los siguientes requerimientos:

- Organización de datos: La aplicación web debe mostrar los datos obtenidos de forma organizada acorde a la placa de cada automotor. El usuario puede elegir que vehículo es el que desea consultar.
- Interpretación de códigos: Con el fin de facilitar el diagnóstico del automóvil a los usuarios, los códigos de falla obtenidos son interpretados por la aplicación Web. Se debe mostrar una descripción sobre el significado de cada código obtenido.
- Implementación de base de datos: Dentro del proceso de desarrollo de la aplicación Web, se debe incorporar una base de datos que permita el almacenamiento de vehículos, códigos de falla, parámetros de funcionamiento y descripción de códigos.
- Presentación de datos: Los parámetros de funcionamiento deben presentarse en el servidor mediante gráficos indicando la fecha y hora de registro de los eventos.

3.1.2 Herramientas y recursos

Automóvil de prueba

Para el desarrollo del sistema se utilizó un vehículo marca Kia, modelo RIO, año 2013. El automóvil mencionado utiliza el protocolo ISO 9141-2 para el manejo de la comunicación OBDII y la computadora del vehículo transmite la información a una tasa de 10.4K Baudios con ocho bits de datos, sin paridad y un bit de parada.

3.2 Recepción de datos desde OBDII

La interacción con el protocolo OBDII se realiza a través del dispositivo BAFX OBDII, el terminal envía una solicitud mediante códigos propios del protocolo en base a valores hexadecimales presentados en formato ASCII. Una vez recibida la petición, el dispositivo responde a la solicitud con una confirmación del tipo de información que va a transmitir, seguido de una trama que contiene los datos requeridos.

De los modos de operación especificados en el Capítulo 2, el desarrollo del proyecto se centrará principalmente en dos de ellos, el modo 01 y el modo 03. Con el modo 1 se puede obtener información instantánea de los parámetros especificados en la sección 3.1.1 del presente capítulo, mientras que con el modo 3 es posible extraer del automóvil los códigos de falla almacenados en la computadora del mismo.

3.2.1 Modo 01

El modo 01 permite el acceso a los datos instantáneos del automotor, sin embargo cumple además con una función indispensable para la comunicación con la computadora del vehículo, dicha función es la inicialización del protocolo OBDII.

En el modo 01 la trama que debe enviarse vía Bluetooth al dispositivo BAFX se compone de dos bytes seguidos de un retorno de carro. El primer byte se encarga de especificar el modo de operación por lo que siempre tendrá el valor 01, al segundo byte se lo conoce como identificador de parámetro o PID por sus siglas en inglés y será el encargado de especificar la acción que se le está solicitando realizar a la computadora del automóvil.

Inicialización de OBDII

Antes de poder extraer cualquier información es necesario inicializar el protocolo enviando la línea “01 00\r”. Ante este comando, el dispositivo envía como respuesta una trama de 6 bytes en la cual el byte 1 y 2 se utilizan como confirmación del modo de operación, mientras los 4 bytes restantes representan los PID soportados por la computadora del automóvil.

Cada bit de los 4 bytes mencionados representa un PID, siendo el PID 1 el bit más significativo de la trama. Si el bit toma el valor de “1” el PID es soportado, caso contrario el bit toma el valor de “0”. En la figura 3.2 se muestra un ejemplo de trama recibida al inicializar el protocolo OBDII.

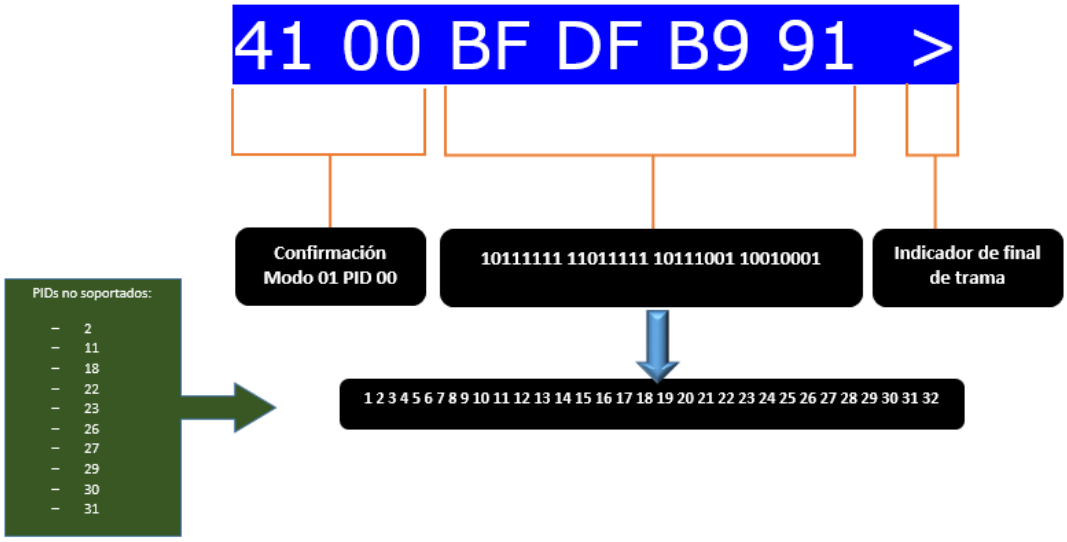


Figura. 3.2 Ejemplo de trama de inicialización OBDII

Para indicar el final de una trama, el protocolo OBDII utiliza el símbolo “>”.

Recepción de parámetros

Una vez inicializado el protocolo OBDII, dentro del modo 01 es posible acceder a los parámetros que se desea monitorear. Para cada parámetro existe un PID asociado que permite solicitar al automóvil su estado en un determinado momento. En la tabla 3.2 se presentan los PIDs asignados a los cinco parámetros con los que se trabajará en la aplicación.

Tabla. 3.2 PIDs asociados

Parámetro	PID
Revoluciones por minuto	0C
Velocidad	0D
Carga Motor	04
Temperatura Refrigerante	05
Posición Acelerador	11

Revoluciones por minuto

Conocer la velocidad a la que gira el motor es de suma importancia al momento de conducir un automóvil, llevar el motor a muy altas o muy bajas revoluciones produce un desgaste considerable en la estructura. A pesar de este hecho algunos fabricantes optan por prescindir del tacómetro (elemento encargado de medir las revoluciones del motor), en varios de sus modelos con el fin de abaratar costos.

Como se observa en la tabla 3.2, el PID necesario para extraer el valor de las revoluciones por minuto a las que gira el motor es “0C”, por lo que el comando que se requiere enviar al dispositivo Bluetooth será “01 0C\r”.

Ante el envío de este comando el vehículo responde con dos tramas, la primera sirve como confirmación del comando recibido y tiene la misma estructura “01 0C\r”. La segunda trama está compuesta de 32 bits, los 16 bits más significativos tendrán siempre el valor “41 0C” que indica que dicha trama es una respuesta al PID “0C” en el modo 01, los restantes 16 bits contienen el valor de las revoluciones del motor en el instante que el comando fue enviado. Al final de la segunda trama se añada el carácter “>” para indicar la finalización de la transmisión de dicho parámetro.

En la figura 3.3 se muestra como se realiza la comunicación entre un dispositivo terminal y el automóvil para obtener las revoluciones por minuto.

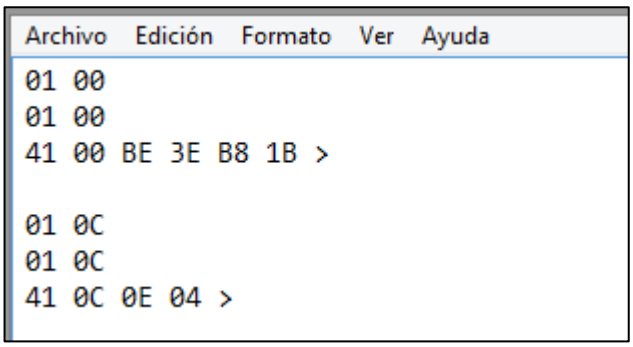


Figura. 3.3 Lectura RPM

El número de revoluciones a las que está girando el motor se obtiene de los 2 bytes menos significativos de la segunda trama, sin embargo es necesario procesar los datos para obtener el valor real del parámetro. Para llegar a dicho valor es necesario aplicar la siguiente fórmula

$$RPM = \frac{(A * 256) + B}{4} \quad (1)$$

Donde B es el byte menos significativo y A es el segundo byte menos significativo. Si se toma el ejemplo mostrado en la figura 3.3, A = 0E y B = 04, al ser valores hexadecimales como primer paso deben ser convertidos a valores decimales:

$$A = 0E_H = 14$$

$$B = 04_H = 4$$

Una vez convertidos, se aplica la fórmula y se obtiene el parámetro buscado.

$$RPM = \frac{(A * 256) + B}{4} = \frac{(14 * 256) + 4}{4} = 897\ RPM \quad (2)$$

Velocidad

La velocidad a la que se desplaza el vehículo es un parámetro de gran relevancia al momento de circular. El conductor debe ser consiente en todo momento de la velocidad a la que se moviliza para de esta forma evitar exceder los límites establecidos a fin de reducir las probabilidades de sanciones y accidentes de tránsito.

El PID asociado a la velocidad del vehículo como se muestra en la tabla 3.2 es el PID “0D”. Por tanto cuando se requiera conocer la velocidad de circulación, el comando a enviar será “01 0D\r”

Ante el envío de este comando el vehículo responde con dos tramas, la primera sirve como confirmación del comando recibido y tiene la misma estructura “01 0D\r”. La segunda trama está compuesta de 24 bits, los 16 bits más significativos tendrán siempre el valor “41 0D” que indica que dicha trama es una respuesta al PID “0D” en el modo 01, los restantes 8 bits contienen la velocidad del automotor en el

instante que el comando fue enviado. Al final de la segunda trama se añada el carácter “>” para indicar la finalización de la transmisión de dicho parámetro.

En la figura 3.4 se muestra como se realiza la comunicación entre un dispositivo terminal y el automóvil para obtener la velocidad del vehículo.

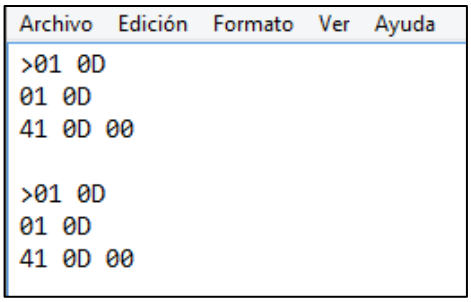


Figura. 3.4 Lectura velocidad

La velocidad del vehículo se obtiene del byte menos significativo de la segunda trama. El byte se recibe en formato hexadecimal, basta con convertir dicho valor a decimal para obtener la velocidad real del automóvil. Para el ejemplo mostrado en la figura 3.4, el byte menos significativo tiene el valor de “00”, esto indica que el automóvil se encuentra detenido.

Carga del motor

El valor de carga del motor es un parámetro poco conocido pero de gran relevancia en el vehículo. La carga indica que tan fuerte es el trabajo al que está sometido el motor. Cuando se habla de carga completa, significa que el acelerador se encuentra totalmente abierto y el motor genera en ese instante toda la potencia posible. Cuando se habla de carga nula el acelerador se encuentra totalmente cerrado, sin embargo esta condición solo se alcanza cuando el vehículo está apagado, dado que aun en ralentí[1], el cuerpo de aceleración se encuentra parcialmente abierto.

La carga del motor es el resultado de la relación de varios parámetros, un motor que se encuentra en una pendiente a 3000 rpm y con el acelerador totalmente abierto

[1] Ralentí: Revoluciones por minuto mínimas a las que un motor de combustión interna requiere girar para mantenerse encendido.

no posee la misma carga de un motor en las mismas condiciones que se moviliza por una zona plana.

El PID asociado a la carga del motor como se muestra en la tabla 3.2 es el PID “04”. Por tanto cuando se requiera conocer la carga del motor, el comando a enviar será “01 04\r”

Ante el envío de este comando el vehículo responde con dos tramas, la primera sirve como confirmación del comando recibido y tiene la misma estructura “01 04\r”. La segunda trama está compuesta de 24 bits, los 16 bits más significativos tendrán siempre el valor “41 04” que indica que dicha trama es una respuesta al PID “04” en el modo 01, los restantes 8 bits contienen la carga del motor en el instante que el comando fue enviado. Al final de la segunda trama se añada el carácter “>” para indicar la finalización de la transmisión de dicho parámetro.

En la figura 3.5 se muestra como se realiza la comunicación entre un dispositivo terminal y el automóvil para obtener la carga del motor.

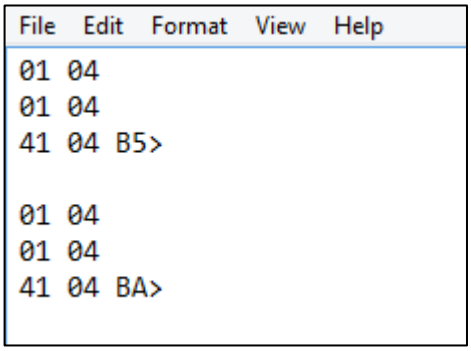


Figura. 3.5 Lectura carga del motor

Para extraer el valor de carga del motor en porcentaje es necesario tomar el byte menos significativo de la segunda trama, convertir este valor de hexadecimal a decimal y aplicar la siguiente fórmula.

$$Carga\ motor(\%) = \frac{A * 100}{255} \tag{3}$$

Donde *A* es el valor del byte menos significativo en formato decimal. Si se toma el ejemplo de la figura 3.5 y se aplica la fórmula se tiene:

$$A=B5_H=181$$

$$Carga\ motor(\%) = \frac{A * 100}{255} = \frac{181 * 100}{255} = 70\% \quad (4)$$

Temperatura del refrigerante

En algunos vehículos es posible encontrar en el tablero de instrumentos un indicador de temperatura que muchas veces se asocia como la temperatura del motor, sin embargo este indicador lo que realmente muestra es la temperatura del líquido refrigerante. Muchos vehículos actualmente no poseen este indicador, lo que representa un serio problema si por alguna razón la temperatura se incrementa. Si se llega a valores superiores a 100° C el motor puede sufrir daños severos.

El PID asociado a la temperatura del refrigerante como se muestra en la tabla 3.2 es el PID "05". Por tanto cuando se requiera conocer la temperatura del refrigerante, el comando a enviar será "01 05\r"

Ante el envío de este comando el vehículo responde con dos tramas, la primera sirve como confirmación del comando recibido y tiene la misma estructura "01 05\r". La segunda trama está compuesta de 24 bits, los 16 bits más significativos tendrán siempre el valor "41 05" que indica que dicha trama es una respuesta al PID "05" en el modo 01, los restantes 8 bits contienen la temperatura en el instante que el comando fue enviado. Al final de la segunda trama se añada el carácter ">" para indicar la finalización de la transmisión de dicho parámetro.

En la figura 3.6 se muestra como se realiza la comunicación entre un dispositivo terminal y el automóvil para obtener la temperatura del refrigerante.

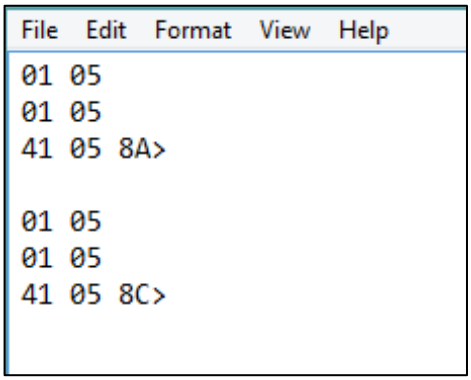


Figura. 2.6 Lectura temperatura del refrigerante

Para obtener la temperatura se aplica la siguiente fórmula:

$$Temperatura\ refrigerante = A - 40 \quad (5)$$

Donde A es el byte menos significativo, convertido de hexadecimal a decimal. Para el ejemplo mostrado en la figura 3.6 se tendría:

$$A=8A_H=138$$

$$Temperatura\ refrigerante = A - 40 = 138 - 40 = 98°C \quad (6)$$

Posición del acelerador

El parámetro posición del acelerador se refiere al porcentaje de abertura del cuerpo de aceleración del vehículo, mas no a la posición del pedal del acelerador, aunque estos dos valores están directamente relacionados.

Junto con la carga del motor este parámetro permite al conductor visualizar de mejor forma el esfuerzo al que se está sometiendo al vehículo y mejorar el estilo de conducción para reducir el desgaste del automóvil y el consumo de combustible.

El PID asociado a la temperatura del refrigerante como se muestra en la tabla 3.2 es el PID "11". Por tanto cuando se requiera conocer la posición del acelerador, el comando a enviar será "01 11\r"

Ante el envío de este comando el vehículo responde con dos tramas, la primera sirve como confirmación del comando recibido y tiene la misma estructura “01 11\r”. La segunda trama está compuesta de 24 bits, los 16 bits más significativos tendrán siempre el valor “41 11” que indica que dicha trama es una respuesta al PID “11” en el modo 01, los restantes 8 bits contienen la posición en el instante que el comando fue enviado. Al final de la segunda trama se añada el carácter “>” para indicar la finalización de la transmisión de dicho parámetro.

En la figura 3.7 se muestra como se realiza la comunicación entre un dispositivo terminal y el automóvil para obtener la posición del acelerador.

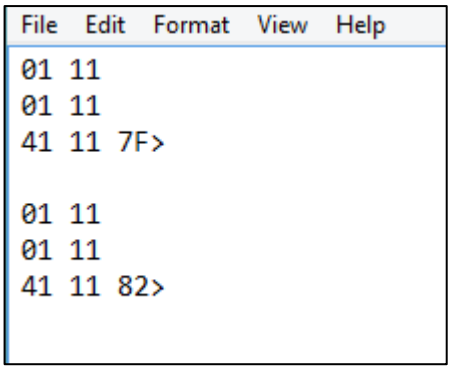


Figura. 3.7 Lectura posición del acelerador

Para extraer la posición del acelerador en porcentaje es necesario tomar el byte menos significativo de la segunda trama, convertir este valor de hexadecimal a decimal y aplicar la siguiente fórmula.

$$Posición\ del\ acelerador(\%) = \frac{A * 100}{255} \qquad (7)$$

Donde A es el valor del byte menos significativo en formato decimal. Si se toma el ejemplo de la figura 3.7 y se aplica la fórmula se tiene:

$$A=7F_H=127$$

$$Posición\ del\ acelerador(\%) = \frac{A * 100}{255} = \frac{127 * 100}{255} = 49\% \qquad (8)$$

3.2.2 Modo 03

El modo 03 realiza un diagnóstico de diversos sistemas del vehículo. Antes de acceder a este diagnóstico es necesario primero haber inicializado el protocolo OBDII como se explica en la sección 3.2.1.1.

A diferencia del modo 01, en el modo 03 no existen PIDs, dado que no es necesario especificar los datos que se desean obtener, basta con enviar hacia el vehículo el comando “03\r” para comenzar el diagnóstico del automóvil.

En el caso de no encontrarse fallas en el vehículo, el mismo responde con una trama que puede tomar tres formatos diferentes de acuerdo al protocolo de comunicación que utilizan. Las tres posibles tramas son:

- NO DATA
- 43 00
- 43 00 00 00 00 00

Si por el contrario se encontrasen códigos de falla en el automóvil, la respuesta al dispositivo será una trama de 7 bytes donde el primer byte siempre tendrá el valor “43”, seguido de 6 bytes de los cuales cada par representa un código de falla. Ejemplos de tramas con códigos de errores son:

- 43 06 85 00 00 00 00
- 43 01 58 02 20 01 18
- 43 02 78 01 54 00 00

En el primer ejemplo solo un par de bytes tienen un valor distinto de cero, por tanto solo existe un código de falla que sería P0685. En el segundo ejemplo todos los bytes son distintos de cero, es decir existen tres códigos de falla reportados, P0158, P0220, P0118 y en el segundo ejemplo de igual manera existen dos códigos de falla P0278 y P0154.

En caso de existir más de tres códigos de falla la respuesta se realizará en múltiples tramas seguidas. Un ejemplo es la trama:

- 43 01 08 01 13 01 18 43 01 22 02 01 02 02 43 02 03 03 51 03 52 43 03 53 11 08 15 52 43 16 14 16 32 00 00

De igual forma los tres pares de bytes siguientes a cada valor “43” representan los códigos de falla presentes en el vehículo.

3.3 Desarrollo de la aplicación móvil

3.3.1 Estructura

La aplicación móvil es el eje fundamental del proyecto, es la encargada de establecer la comunicación Bluetooth con el automóvil, procesar las tramas recibidas y enviar los datos al servidor a través de internet. La aplicación se desarrolló para ser compatible con dispositivos Android 2.3 en adelante. El diagrama de bloques del sistema se muestra en la figura 3.8.

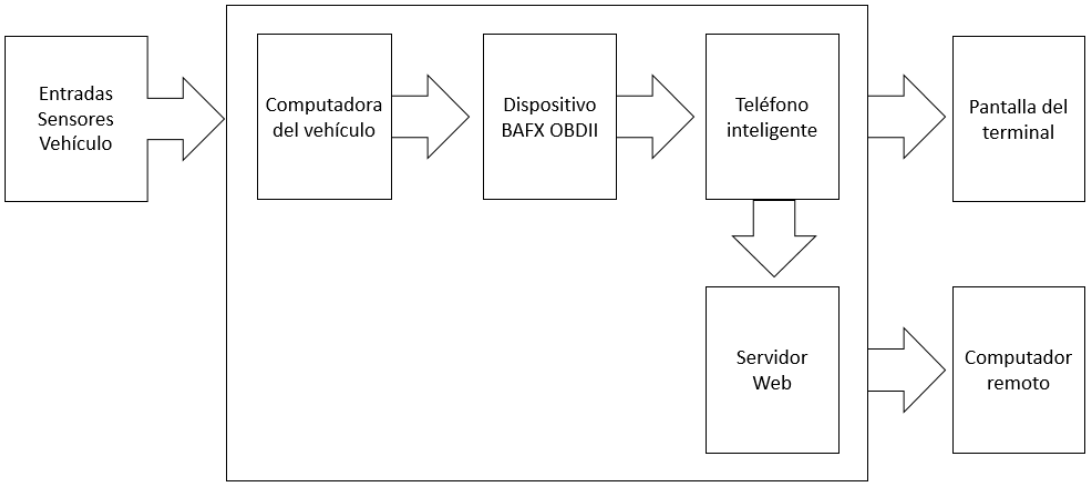


Figura. 3.8 Diagrama de bloques del sistema

Diagramas UML

El diagrama UML de casos de uso del sistema se muestra en la figura 3.9. Los actores presentes en el diagrama son el conductor del vehículo, la aplicación móvil y el servidor que presentará la información de forma remota. El conductor se encarga de realizar el emparejamiento Bluetooth con el dispositivo BAFX. Para esto primero

la aplicación móvil debe realizar la búsqueda del dispositivo, una vez conectado, se decide el modo de operación del sistema.

Si se selecciona el modo de diagnóstico, la aplicación móvil escanea la computadora del vehículo en busca de fallas, la información se envía al servidor y se almacena en la base de datos. La aplicación web selecciona el vehículo que desea revisar, se conecta a la base de datos, extrae los códigos de falla, los interpreta y finalmente los presenta a cualquier usuario que este accediendo a la aplicación web.

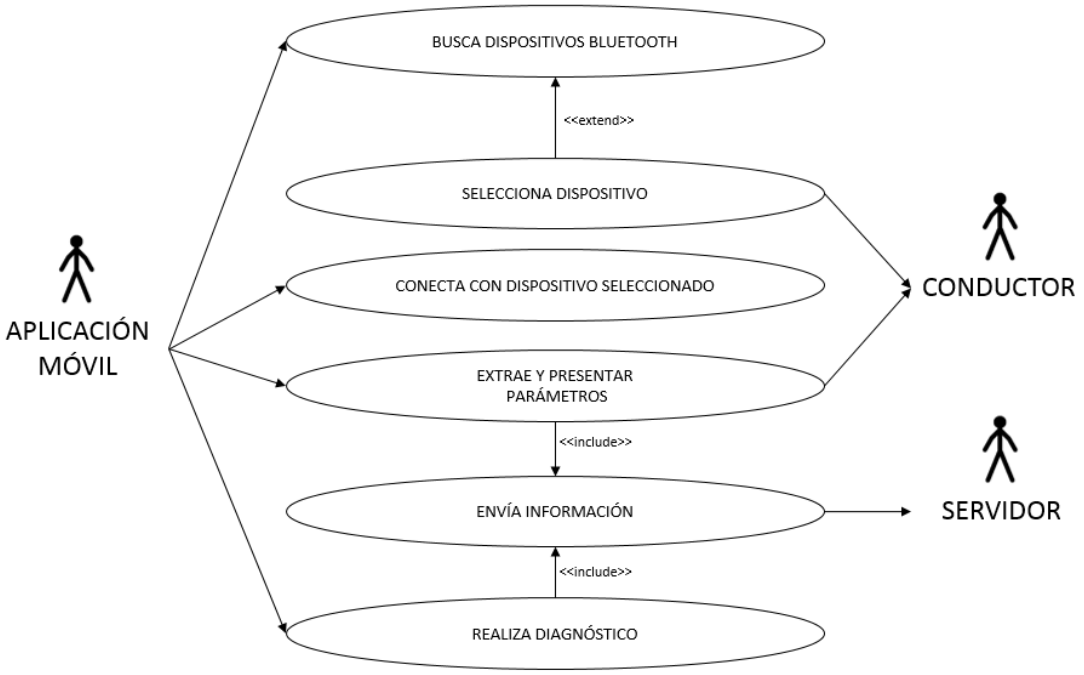


Figura. 3.9 Diagrama UML de casos de uso del sistema

Si se elige el modo de monitoreo local, la información es presentada al conductor en la pantalla de teléfono y enviada al servidor en un intervalo de un minuto entre cada envío.

La aplicación móvil desarrollada se compone de once clases que serán detalladas en las siguientes secciones. El diagrama UML que muestra las clases y las relaciones entre ellas se muestran en la figura 3.10.

Figura. 3.10 Diagrama UML de Clases

3.3.2 Interfaz de usuario

Para lograr que el desempeño del sistema sea óptimo, se estableció tres criterios de diseño principales que debe cumplir la interfaz de usuario de la aplicación. Estos criterios son:

- Interfaz intuitiva.
- De fácil comprensión.
- Coherente con la información mostrada.

La conectividad Bluetooth es imprescindible en la aplicación por tanto si la comunicación Bluetooth en el terminal se encuentra desactivada, al iniciar la aplicación se presentará una ventana solicitando la activación de la comunicación Bluetooth. En la figura 3.11 se muestra el diseño de la ventana que corresponde a esta acción, mientras en la figura 3.12 se muestra la ventana ya implementada.

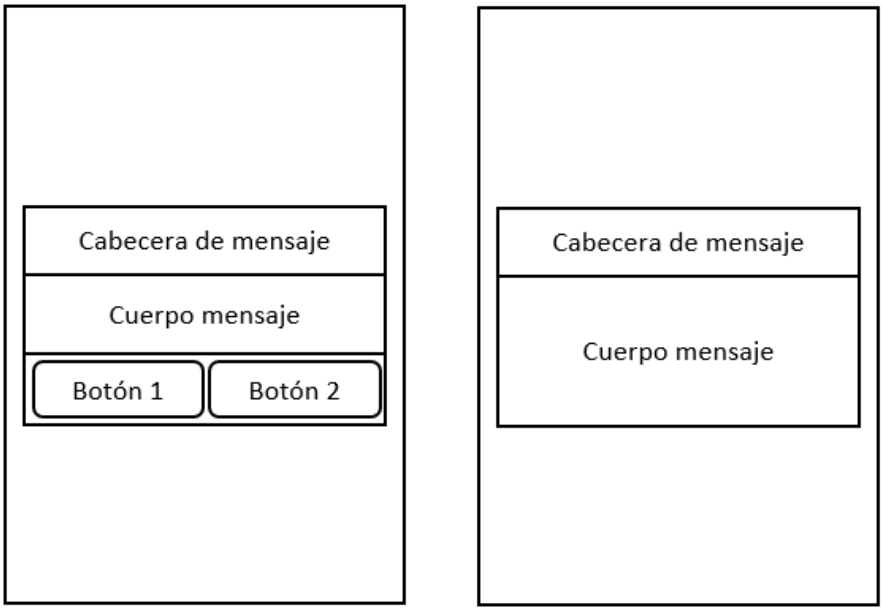


Figura. 3.11 Diseño ventana de activación Bluetooth

Figura. 3.12 Mensaje de activación Bluetooth

Dado que está relacionado con el manejo del vehículo, el monitoreo local es el modo de operación predominante y la ventana que muestra esta información, es la ventana principal en la aplicación. En la figura 3.13 se muestra el diseño de la ventana principal con sus componentes y en la figura 3.14 se presenta la implementación de la ventana en la aplicación.

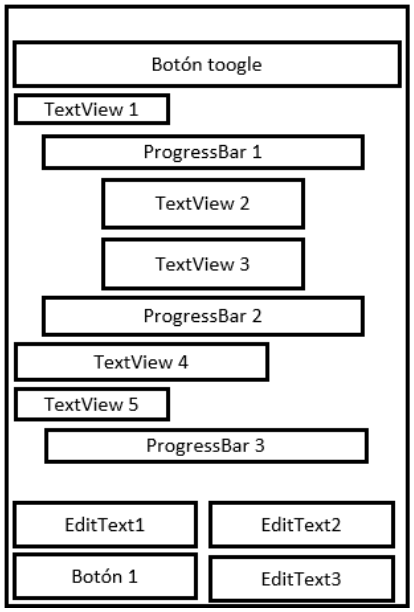


Figura. 3.13 Diseño ventana principal

Figura. 3.14 Ventana principal y opción de conexión Bluetooth

En la tabla 3.3 se muestra la descripción de cada componente de la ventana principal.

Tabla. 3.3 Componentes ventana principal

Componente	Descripción
Botón toogle	Permite la activación o desactivación de la lectura de datos.
Botón 1	Cambiar al modo de diagnóstico.
EditText1	Se utiliza para el ingreso de la Placa del automóvil
EditText2	Se utiliza para el ingreso del Modelo del automóvil
EditText3	Se utiliza para el ingreso de la Marca del automóvil
TextView 1	Muestra la temperatura del refrigerante.
TextView 2	Muestra la velocidad del vehículo.
TextView 3	Muestra las revoluciones por minuto del motor.
TextView 4	Muestra la carga del motor.
TextView 5	Muestra la posición del acelerador.
ProgressBar 1	Muestra la temperatura del refrigerante en una animación.
ProgressBar 2	Muestra las revoluciones del motor en una animación.
ProgressBar 3	Muestra la posición del acelerador en una animación.

Al iniciar la aplicación, la ventana se presentará ante el usuario pero sin mostrar ninguna información. Es necesario primero conectar el teléfono al dispositivo OBDII para que los datos se transmitan. Para esto se diseñó que al presionar sobre el ícono de opciones propio del teléfono aparece el botón “Conectar al adaptador OBDII”.

Al presionar la opción mencionada se despliega una nueva ventana en donde se pueden visualizar los dispositivos Bluetooth anclados al teléfono y un botón mediante el cual se puede escanear el entorno en busca de nuevos dispositivos. En la figura 3.15 se muestra el diseño de la ventana mencionada y en la figura 3.16 la implementación de la ventana.

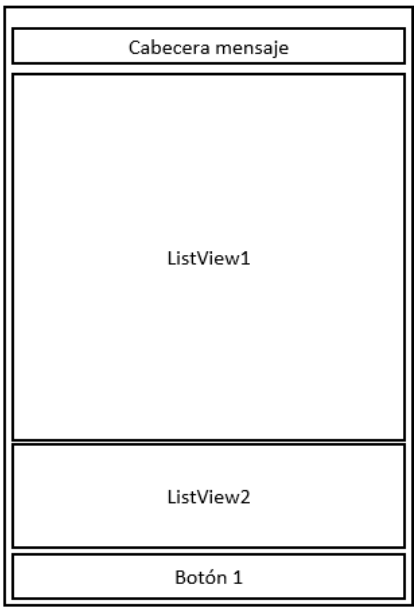


Figura. 3.15 Diseño ventana búsqueda de dispositivos

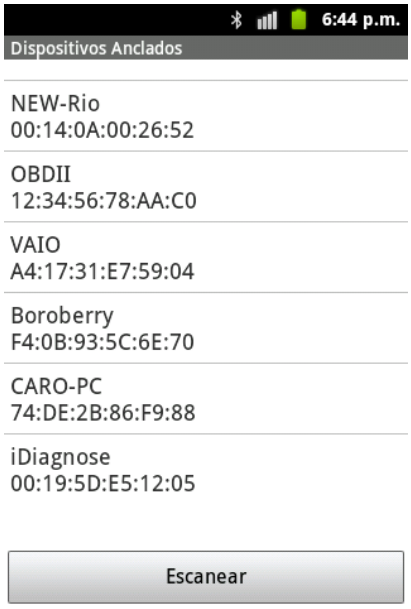


Figura. 3.16 Ventana de búsqueda de dispositivos

En la tabla 3.4 se presenta la descripción de los componentes de la ventana de búsqueda de dispositivos.

Tabla. 3.4 Componentes de ventana de búsqueda

Componente	Descripción
Cabecera mensaje	Muestra mensaje con título de la lista
ListView1	Lista de dispositivos emparejados
ListView2	Lista de dispositivos nuevos encontrados
Botón 1	Botón realizar búsqueda

Cuando se ha establecido ya la conexión Bluetooth con un dispositivo es posible comenzar el monitoreo local o el diagnóstico del vehículo.

En el modo de operación de diagnóstico se presenta una ventana al conductor donde puede realizar el escaneo del vehículo en busca de fallas y a su vez mediante un botón enviar, almacenar estos datos en el servidor. El diseño de la ventana se muestra en la figura 3.17 mientras la ventana implementada se presenta en la figura 3.18.

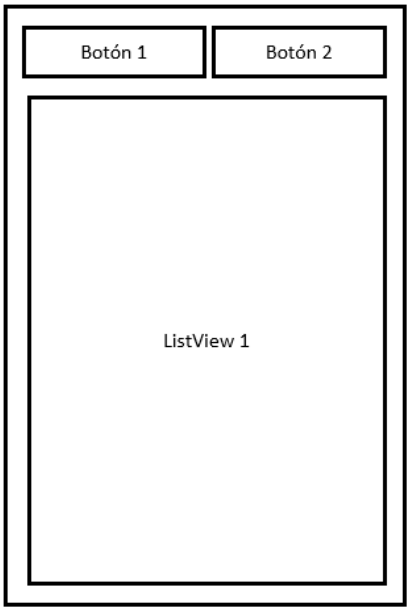


Figura. 3.17 Diseño ventana en modo diagnóstico

Figura. 3.18 Ventana en modo de diagnóstico

En la tabla 3.5 se presenta la descripción de los componentes de la ventana de diagnóstico.

Tabla. 3.5 Componentes ventana en modo de diagnóstico

Componente	Descripción
Botón 1	Realiza el diagnóstico
Botón 2	Envía resultados al servidor
ListView 1	Muestra lista de códigos de falla encontrados

3.3.3 Comunicación Bluetooth

Dentro del diseño de la aplicación, el punto de partida para el sistema es la conexión entre el dispositivo Bluetooth y el teléfono. En base a esta consideración, se realiza una validación previa al inicio de la aplicación. Si el dispositivo en el cual está instalada la aplicación no posee físicamente el soporte para conexiones Bluetooth, la aplicación se cierra automáticamente para no ocasionar conflictos en el terminal. Caso contrario la aplicación presenta en pantalla un mensaje solicitando la activación Bluetooth. Si se confirma la activación, la aplicación accede a la pantalla principal, si por el contrario no se confirma, la aplicación se cierra automáticamente.

Clase ServicioBluetoothChat

Esta clase realiza todo el trabajo necesario para establecer y gestionar la comunicación Bluetooth en el proyecto. Tiene un hilo que escucha en espera de conexiones entrantes, un hilo encargado de realizar la conexión con otro dispositivo y un hilo que realiza la transmisión de datos cuando ya se estableció la conexión.

Los métodos implementados en la clase se listan y explican en la tabla 3.6.

Tabla. 3.6 Clase ServicioBluetoothChat

Tipo	Método	Descripción
constructor	ServicioBluetoothChat(Context contexto, Handler handler)	Constructor de la clase, recibe como argumentos la actividad Principal y el Handler
void	establecerEstado(int estado)	Establece el estado actual de la conexión
int	obtenerEstado()	Retorna el valor del estado actual de la conexión
void	comenzar()	Comenzar el hilo de escucha en espera de conexiones entrantes
void	conectar(BluetoothDevice dispositivo, boolean seguro)	Comenzar el hilo de conexión con otro dispositivo. Se recibe como argumento el dispositivo a conectar y un boolean que identifica el tipo de conexión
void	conectado(BluetoothSocket socket, BluetoothDevice dispositivo)	Comenzar el hilo que maneja la conexión Bluetooth. Se recibe como argumento el socket de la conexión, el dispositivo Bluetooth ya conectado y el tipo de socket que se utiliza
void	detener()	Detener todos los hilos que se encuentren corriendo
void	escribir(byte[] salida)	Salida de datos una vez realizada la conexión
void	conexionFallida()	Devuelve un mensaje si la conexión no pudo realizarse
void	conexionPerdida()	Devuelve un mensaje si la conexión se cayó o se interrumpió
void	cancela()	Cierra el socket y la comunicación

Clase HiloConectado

Este hilo se encarga de manejar la conexión cuando el terminal ha logrado conectarse con un dispositivo Bluetooth. Gestiona además el envío y recepción de datos desde y hacia el teléfono para que puedan ser procesados en la clase Principal.

Los métodos implementados en la clase y su respectiva descripción se encuentran especificados en la tabla 3.7.

Tabla. 3.7 Clase HiloConectado

Tipo	Método	Descripción
constructor	HiloConectado(BluetoothSocket socket, String tipoSocket)	Constructor de la clase, se recibe como argumentos el socket de la conexión y el tipo de socket
void	run()	Recibe los parámetros desde el dispositivo Bluetooth y los envía a la clase Principal
void	escribir(byte [] buffer)	Envía los parámetros al dispositivo Bluetooth y notifica a la clase Principal
void	cancela()	Cierra el socket y la comunicación

3.3.4 Lectura y procesamiento de datos OBDII

Dentro de los procesos manejados en el proyecto, la lectura de datos provenientes del automóvil es uno de los aspectos críticos del desarrollo. Con el fin de elaborar un algoritmo adecuado y eficiente, antes de realizar el desarrollo, varias pruebas con el software Tera Term fueron llevadas a cabo. En estas pruebas se recopiló información sobre la estructura de las tramas que el vehículo enviaba como respuesta a los comandos enviados. Estas tramas se presentaron en la sección 3.2.

Dado que el vehículo solo envía datos cuando recibe una solicitud del teléfono, los comandos deben ser enviados regularmente hacia el automóvil para mantener

constante el monitoreo de los parámetros. Por tanto en el modo de monitoreo local la aplicación entrará en un bucle constante a menos que el usuario cambie al modo de diagnóstico.

Cuando se realizaron las pruebas, los comandos se enviaron manualmente, esto brindaba al dispositivo BAFX la suficiente cantidad de tiempo para responder con la trama en el formato correcto, sin embargo al enviar los cinco comandos correspondientes a los parámetros que se desean monitorear, las tramas enviadas desde el dispositivo BAFX contenían errores o caracteres basura que volvían difícil la interpretación de datos. Este problema se debe principalmente a que la velocidad con la que el terminal envía los comandos es superior a la velocidad de respuesta del dispositivo.

En este punto es donde entra el uso de expresiones regulares. Se identificaron mediante pruebas los formatos de trama que el dispositivo enviaba cuando se le sometía a una consulta constante de parámetros. Estos formatos se presentan en la tabla 3.8.

Tabla. 3.8 Formatos de trama en monitoreo local

Formato	Descripción
41 X_1 X_2 X_3	Solo utilizado para RPM. X_1=0C, X_2 - X_3= Valores de interés.
41 X_1 X_2	Utilizado para Carga del motor, Temperatura, Posición del acelerador, Velocidad. X_1=0D o X_1=04 o X_1=05 o X_1=11, X_2=Valor de interés
X_1 X_2	Utilizado para RPM X_1 – X_2 = Valores de interés

Una vez identificados los tres formatos de trama más frecuentes, se implementaron expresiones regulares que permiten a la aplicación filtrar las tramas provenientes del vehículo. Las expresiones regulares establecidas que arrojan coincidencia con las tramas definidas como recurrentes se presentan en la tabla 3.9.

Tabla. 3.9 Expresiones regulares implementadas

Trama	Expresión regular
41 X1 X2 X3	\\s*[0-9A-Fa-f]{2} [0-9A-Fa-f]{2} [0-9A-Fa-f]{2} [0-9A-Fa-f]{2}\\s*\r?\n?
41 X1 X2 X3 >	\\s*[0-9A-Fa-f]{2} [0-9A-Fa-f]{2} [0-9A-Fa-f]{2} [0-9A-Fa-f]{2} \\s*>\\s*\r?\n?
41 X1 X2	\\s*[0-9A-Fa-f]{1,2} [0-9A-Fa-f]{2} [0-9A-Fa-f]{2}\\s*\r?\n?
41 X1 X2 >	\\s*[0-9A-Fa-f]{1,2} [0-9A-Fa-f]{2} [0-9A-Fa-f]{2} \\s*>\\s*\r*\n*
X1 X2	\\s*[0-9A-Fa-f]{2} [0-9A-Fa-f]{2}\\s*\r*\n*
X1 X2 >	\\s*[0-9A-Fa-f]{2} [0-9A-Fa-f]{2} \\s*>\\s*\r*\n*
>	\\s*[.A-Za-z0-9\\?*>\r\n]*\\s*>\\s*\r*\n*

En la mayoría de expresiones regulares implementadas, se encuentra la expresión:

- [0-9A-Fa-f]{2}

Esta expresión identifica los bytes enviados desde el dispositivo Bluetooth que se encuentra conectado al vehículo. Se marcará como coincidencia a toda cadena de texto que contenga 2 caracteres, los cuales pueden ser: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, A, B, C, D, E, F.

Gracias a la expresión [0-9A-Fa-f]{2} se puede conformar las expresiones regulares que identificarán a las tramas de interés. En las figuras de la 3.19 a la 3.24 se muestra la representación de las tramas mediante expresiones regulares.

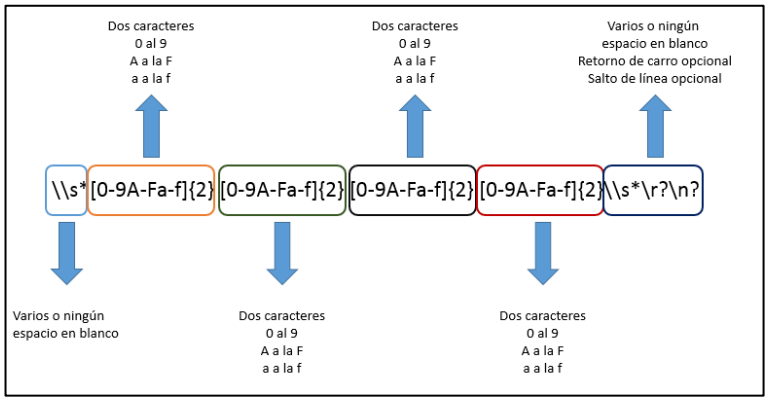


Figura. 3.19 Expresión regular para trama 1

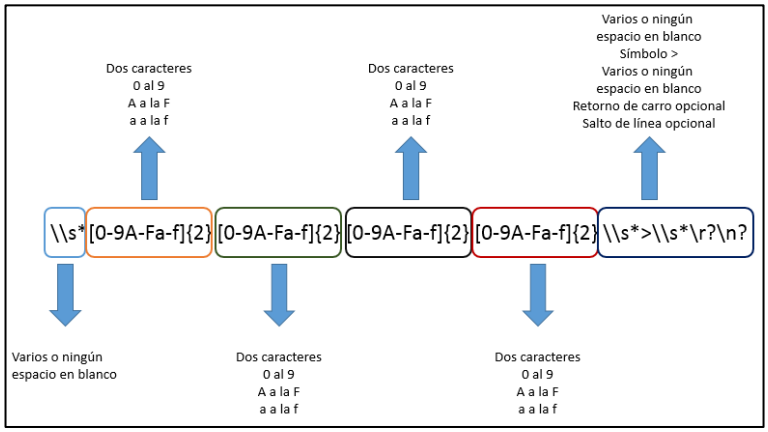


Figura. 3.20 Expresión regular para trama 2

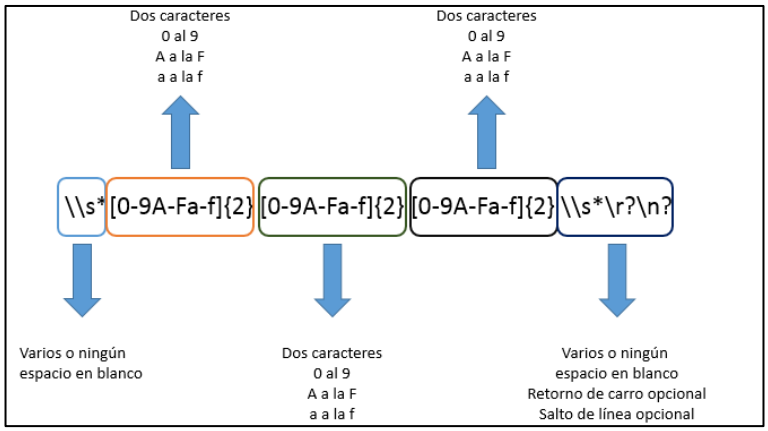


Figura. 3.21 Expresión regular para trama 3

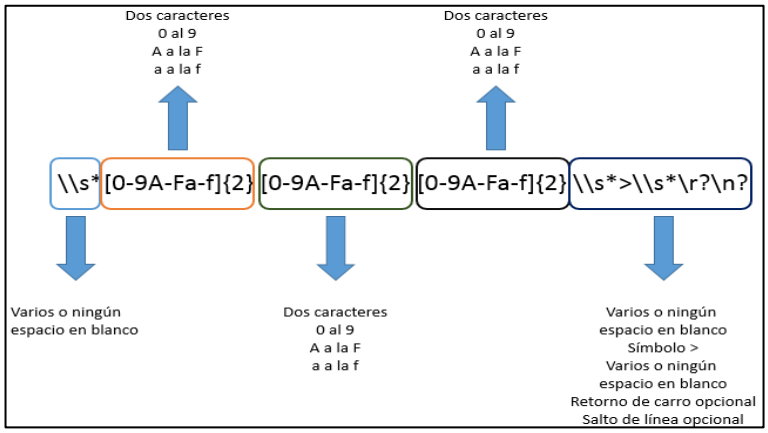


Figura. 3.22 Expresión regular para trama 4

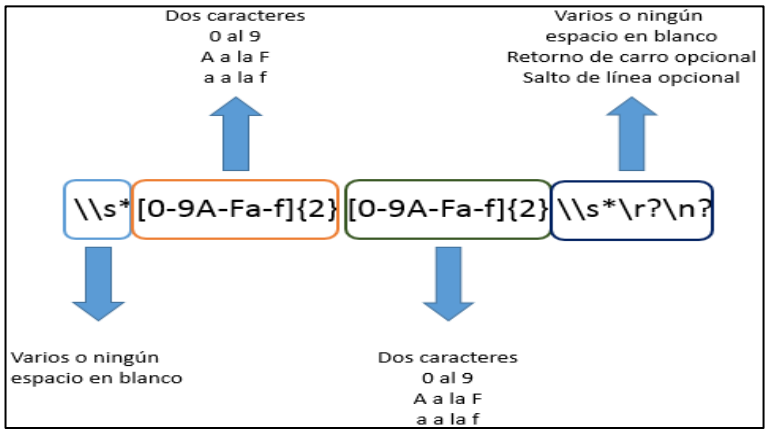


Figura. 3.23 Expresión regular para trama 5

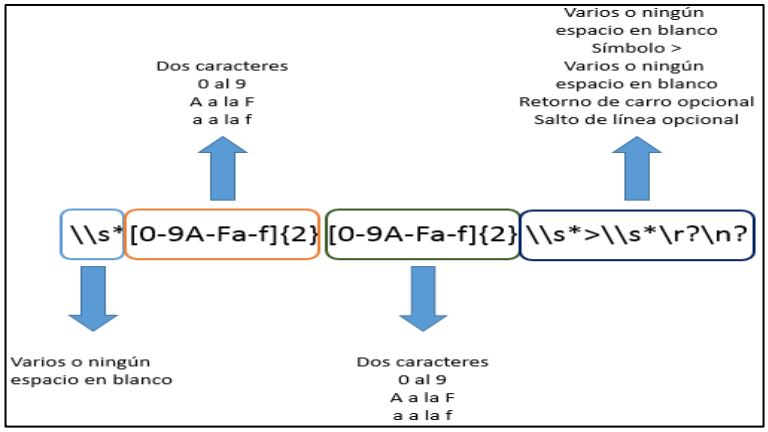


Figura. 3.24 Expresión regular para trama 6

El algoritmo diseñado para realizar el proceso de lectura de tramas de los cinco parámetros de interés se representa mediante el diagrama de flujo mostrado en la figura 3.25.

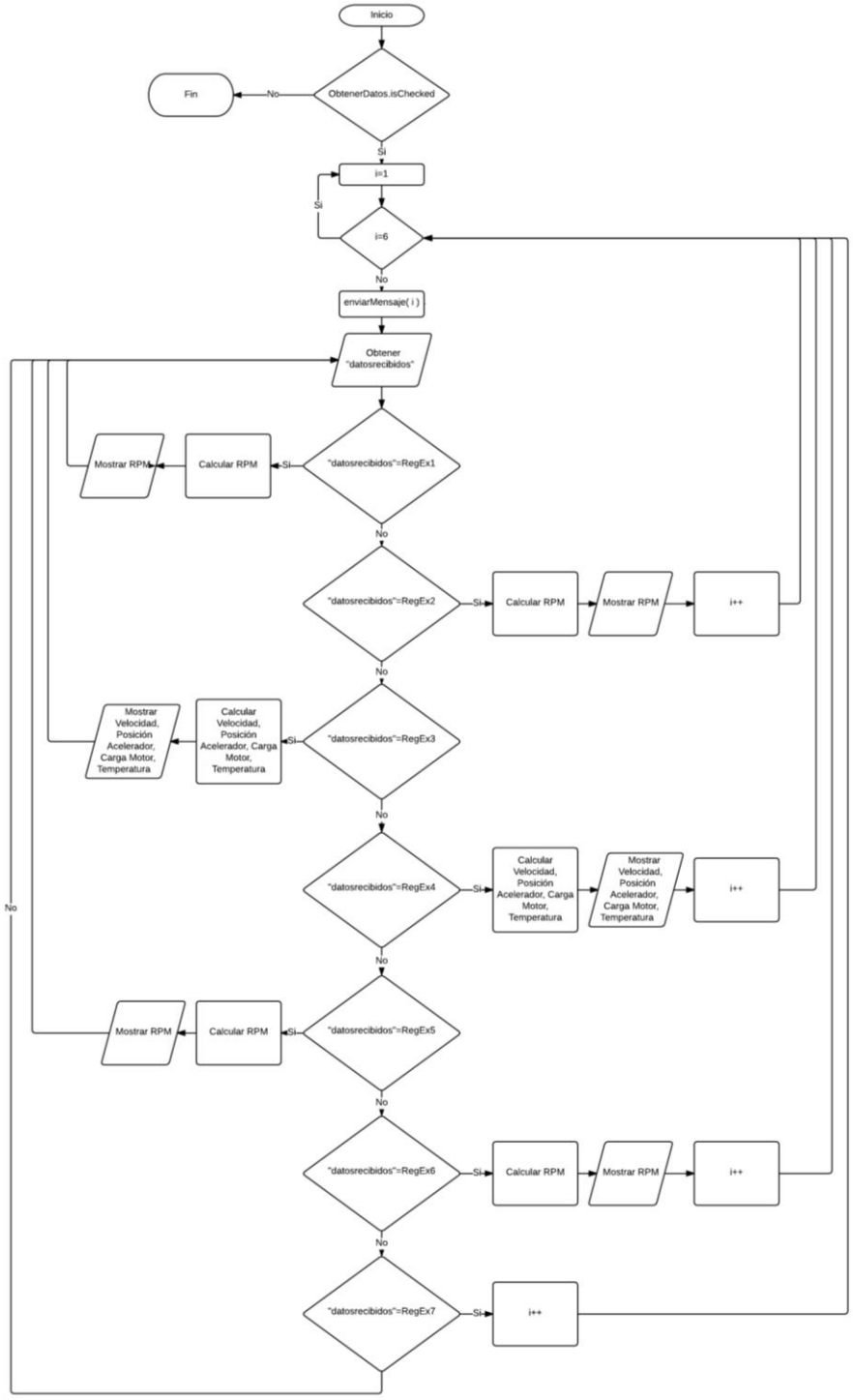


Figura. 3.25 Diagrama de flujo para lectura de tramas OBDII

Clase Principal

La clase Principal es la encargada de procesar los datos recolectados del vehículo en el modo de monitoreo local. En esta clase se realiza toda la interpretación y conversión de tramas. Uno de los aspectos más desafiantes en la realización del proyecto es justamente la conversión de los datos obtenidos desde texto a valores procesables y la discriminación correcta de los datos válidos de los no utilizables.

Es en esta clase donde se implementa el uso de las expresiones regulares establecidas. Mediante sentencias condicionales se permite que la aplicación tome únicamente las tramas que posean el formato mostrado en la tabla 3.9. Caso contrario son descartadas.

Si una trama coincide con una de las expresiones regulares, la trama está lista para ser procesada, sin embargo la trama es una cadena de caracteres. Debe ser primero convertida a valores hexadecimales.

La conversión se realiza utilizando dos métodos de la clase String. Los métodos Trim y Split. El método trim permite eliminar los espacios en blanco al inicio y al final de la cadena. Con esto se asegura que los únicos espacios en blanco que deben filtrarse se encuentran entre los caracteres de la trama.

Como se observa en la tabla 3.8, los bytes de las tramas de interés se encuentran divididos por un espacio en blanco. La eliminación de estos espacios y la separación de la cadena en un arreglo de cadenas se realizan mediante el método Split. Split divide una cadena de caracteres cuando encuentra una expresión coincidente con la expresión enviada como argumento, para el caso del proyecto el argumento enviado es un espacio en blanco. Con esto se asegura que cada ítem del arreglo de cadenas corresponda a los bytes enviados por el vehículo.

El siguiente paso es la conversión de cada ítem del arreglo de cadenas a un valor hexadecimal, para esto se hizo uso del método parseInt de la clase Integer. Usualmente cuando se utiliza este método enviando como argumento únicamente la cadena que se desea convertir, el resultado es la conversión de la cadena a un valor decimal. Sin embargo, si como argumentos además de la cadena que se desea

convertir, se envía un número indicando la base del sistema. La conversión de la cadena se realizará en el sistema deseado. Para el proyecto se envía como argumentos cada ítem del arreglo obtenido y el número 16 indicando que se desea convertir la cadena a un valor hexadecimal. En la figura 3.26 se ilustra el proceso de filtrado y conversión de una cadena en un ejemplo.

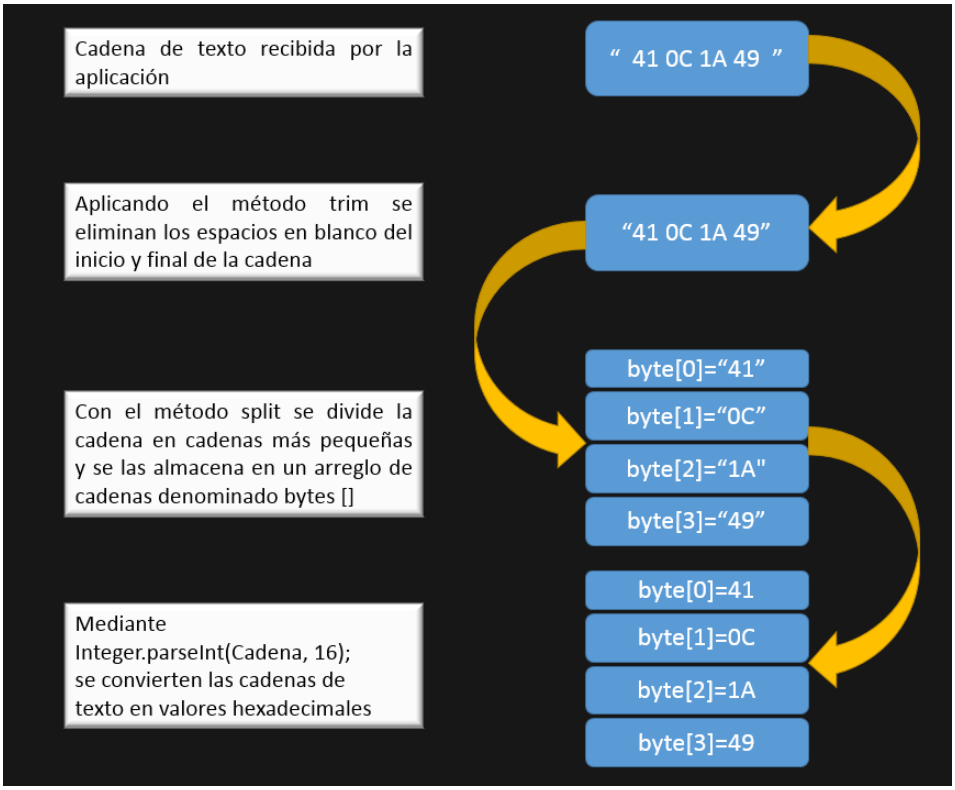


Figura. 3.26 Ejemplo de conversión de tramas

La actividad Principal además de realizar el proceso de lectura e interpretación de tramas, provee el acceso a la ventana de búsqueda y conexión Bluetooth, así como a la ventana de diagnóstico del vehículo. Además envía al servidor cada minuto una captura de los datos monitoreados en dicho instante y registra al vehículo en la base de datos.

En la tabla 3.10 se presentan los métodos implementados en la clase Principal con su respectiva descripción.

Tabla. 3.10 Clase Principal

Tipo	Método	Descripción
void	onCreate(Bundle savedInstanceState)	Define la ventana correspondiente a la actividad
void	onStart()	Si Bluetooth está desactivado, solicita encenderlo
void	onPause()	Muestra un mensaje en consola de encontrarse la aplicación en pausa
void	onStop()	Muestra un mensaje en consola de encontrarse la aplicación detenida
void	onDestroy()	Detiene la transmisión Bluetooth.
void	onResume()	Si Bluetooth está desactivado en onStart(), la aplicación entra a onPause, al terminar la activación regresa a onResume(). Permite continuidad en la aplicación
void	configurarChat()	Inicializa los objetos necesarios para la comunicación Bluetooth. Además especifica las acciones de los botones para activar monitoreo y realizar diagnóstico
void	comenzarTransmision()	Inicializa el protocolo OBDII enviando la trama "01 00\r"
void	getData(int numeromensaje)	Define el PID que se requiere enviar en base al argumento numeromensaje. numeromensaje es un entero de 1 a 5 el cual representa a cada uno de los PIDs solicitados en el monitoreo
void	enviarMensaje(String mensaje)	Envía las tramas definidas en getData(int numeromensaje) al dispositivo Bluetooth

boolean	onOptionsItemSelected(MenuItem item)	Lanza la ventana que permite la búsqueda y conexión de dispositivos Bluetooth, retorna TRUE si la actividad se lanzó correctamente
boolean	onCreateOptionsMenu(Menu menu)	Define la ventana correspondiente al menú de selección de dispositivos
void	onActivityResult(int requestCode, int resultCode, Intent data)	Maneja los casos cuando la actividad de búsqueda y selección de dispositivos retorna un dispositivo para conectar, no retorna nada o se requiere activación de Bluetooth
void	conectarDispositivo(Intent data, boolean secure)	Define la dirección MAC del dispositivo al que se quiere conectar y la entrega a un objeto de la clase ServicioBluetoothChat para que gestione la conexión

La lectura, conversión, interpretación de tramas y comunicación con el servidor se realiza en el Handler establecido en la case Principal, por tanto dentro de la lista de métodos implementados en la clase no consta un método que realice como tal el procesamiento de las tramas y envío de datos.

Clase ActividadCodigoFalla

La clase ActividadCodigoFalla es la encargada de manejar e interpretar las tramas extraídas del vehículo en el modo de diagnóstico. A diferencia de la clase Principal, el comando enviado para obtener la información necesaria no se envía de forma iterativa, sino por el contrario solo cuando el conductor del vehículo presiona el botón correspondiente.

El método de conversión y manipulación de las tramas es similar al utilizado en la clase Principal. La diferencia más relevante son las expresiones regulares determinadas para el modo de diagnóstico frente a las utilizadas en el modo de monitoreo local.

Las expresiones regulares utilizadas cubren tres casos posibles. Cuando no existen códigos de falla en el vehículo, cuando existen hasta 3 códigos de falla y cuando existen más de 3 códigos de falla. Las expresiones usadas en el modo de diagnóstico para cubrir los tres casos mencionados se presentan en la tabla 3.11.

Tabla. 3.11 Expresiones regulares implementadas

Trama	Expresión regular
NO DATA	\\s*[Nn]{1}[Oo]{1} [Dd]{1}[Aa]{1}[Tt]{1}[Aa]{1}\\s*\r?\n?
43 X1 X2 X3 X4 X5 X6	("\\s*[0-9A-Fa-f]{2} [0-9A-Fa-f]{2} [0-9A-Fa-f]{2} [0-9A-Fa-f]{2} [0-9A-Fa-f]{2} [0-9A-Fa-f]{2} [0-9A-Fa-f]{2}\\s*\r?\n?"
43 X1 X2 X3	("\\s*[0-9A-Fa-f]{2} [0-9A-Fa-f]{2} [0-9A-Fa-f]{2} [0-9A-Fa-f]{2}\\s*\r?\n?"
X1 X2	("\\s*[0-9A-Fa-f]{2} [0-9A-Fa-f]{2}\\s*\r?\n?"
>	\\s*[.A-Za-z0-9\\?*>\r\n]*\\s*>\\s*\r*\n*

El proceso es exactamente igual al desarrollado en la clase Principal. La depuración y conversión de las tramas se realiza de forma idéntica con la única diferencia que se añade una expresión regular que arroje coincidencias con las cadenas de texto “NO DATA” y “no data” para el caso cuando el vehículo no posea códigos de falla.

Dado que el vehículo de prueba solo trabaja con códigos de falla en el tren motriz. Todos los códigos arrojados por la computadora tendrán como primer carácter la letra ‘P’. Los códigos de fallas encontrados se listan y presentan en la

pantalla del terminal. La clase ActividadCodigoFalla posee los métodos mostrados en la tabla 3.12.

Tabla. 3.12 Clase ActividadCodigoFalla

Tipo	Método	Descripción
void	onCreate(Bundle savedInstanceState)	Define la ventana correspondiente a la actividad
void	onStart()	Si Bluetooth está desactivado, solicita encenderlo
void	onPause()	Muestra un mensaje en consola de encontrarse la aplicación en pausa
void	onStop()	Muestra un mensaje en consola de encontrarse la aplicación detenida
void	onDestroy()	Detiene la transmisión Bluetooth.
void	onResume()	Si Bluetooth está desactivado en onStart(), la aplicación entra a onPause, al terminar la activación regresa a onResume(). Permite continuidad en la aplicación
void	configurarChat()	Inicializa los objetos necesarios para la comunicación Bluetooth.
void	comenzarTransmision()	Inicializa el protocolo OBDII enviando la trama "01 00\r"
void	getData(int numeromensaje)	Envía el comando "03\r" al dispositivo Bluetooth
void	enviarMensaje(String mensaje)	Envía la trama definida en getData(int numeromensaje) al dispositivo Bluetooth
void	void conectarDispositivo(Intent data, boolean secure)	Mantiene la conexión enviada desde la clase Principal

Clase ListaDispositivos

La clase ListaDispositivos maneja la ventana en la cual el usuario puede escanear el entorno en busca de nuevos terminales Bluetooth, conocer los dispositivos emparejados al teléfono y seleccionar el dispositivo con el que se desea conectar.

Una vez seleccionado el dispositivo, la aplicación regresa a la actividad principal pero tiene ya la información necesaria para establecer la conexión con el dispositivo deseado. Los métodos utilizados en la clase se presentan en la tabla 3.13.

Tabla. 3.13 Clase ListaDispositivos

Tipo	Método	Descripción
void	onCreate(Bundle savedInstanceState)	Define la ventana correspondiente a la actividad
void	onDestroy()	Terminar la búsqueda de dispositivos
void	realizarBusqueda()	Comienza la búsqueda de nuevos dispositivos

Dentro de la inicialización de un objeto del tipo OnItemClickListener se realiza la selección del dispositivo Bluetooth buscado. Se extrae la dirección MAC requerida y se transfiere este valor a la actividad Principal.

3.3.5 Comunicación con el servidor

Una vez realizado el diagnóstico en busca de fallos, el conductor del vehículo puede enviar los códigos recopilados a un servidor. Para esta tarea la aplicación se soporta en dos clases, la clase HttpCliente y la clase Error.

Para almacenar en el servidor los datos correspondientes al monitoreo local, la aplicación hace uso de las clases Parámetro y HttpClientePar. De similar manera, para el registro de automóviles se utilizan las clases Auto y HttpClienteAuto.

Clase Error

La clase Error define la estructura de los datos que se enviarán al servidor de acuerdo a la base de datos creada para el almacenamiento de los códigos de falla de la aplicación. La clase se encarga de determinar a qué automóvil corresponde un código de falla registrado en base a la placa del vehículo.

Los métodos que componen la clase Error se muestran en la tabla 3.14.

Tabla. 3.14 Clase Error

Tipo	Método	Descripción
int	getIdError()	Retorna el identificador del código registrado
void	setIdError(int iderror)	Define el valor del identificador de código en caso de necesitarse
String	getCodigo()	Retorna el código de falla registrado
void	setCodigo(String codigo)	Define el código de falla
String	getDescripcion()	Retorna la descripción del código
void	setDescripcion(String descripcion)	Define la descripción del código en caso de necesitarse
String	getPlaca()	Retorna la placa del vehículo
void	setPlaca(String placa)	Define la placa del vehículo
void	guardarError(Activity actividad)	Mediante un objeto HttpCliente inserta los valores requeridos en la base de datos

Clase Auto

La clase Auto se encarga de definir al vehículo que está siendo diagnosticado, es decir define los parámetros Placa, Marca y Modelo que serán enviados a la tabla Auto de la base de datos diseñada.

Los métodos que componen la clase Auto se presentan en la tabla 3.15.

Tabla. 3.15 Clase Auto

Tipo	Método	Descripción
String	getPlaca()	Retorna la placa del vehículo
void	setPlaca(String placa)	Define la placa del vehículo
String	getMarca()	Retorna la marca del vehículo
void	setMarca(String marca)	Define la marca del vehículo
String	getModelo()	Retorna el modelo del vehículo
void	setModelo(String modelo)	Define el modelo del vehículo
void	guardarAuto(Activity actividad)	Mediante un objeto HttpClienteAuto inserta los valores requeridos en la base de datos

Clase Parámetro

La clase Parámetro se encarga de definir al conjunto de valores correspondientes a los parámetros de monitoreo local. Es decir Parámetro engloba a los valores de la Velocidad, RPM, Carga del motor, Temperatura de refrigerante y Posición del acelerador en cada minuto del recorrido.

Los métodos que componen la clase Auto se presentan en la tabla 3.16.

Tabla. 3.16 Clase Parámetro

Tipo	Método	Descripción
String	getPlaca()	Retorna la placa del vehículo
void	setPlaca(String placa)	Define la placa del vehículo
String	getRpm()	Retorna las rpm del vehículo
void	setRpm(String rpm)	Define las rpm del vehículo (De ser necesario)
String	getVelocidad()	Retorna la velocidad del vehículo
void	setVelocidad(String velocidad)	Define la velocidad del vehículo (De ser necesario)
String	getTemperatura()	Retorna la temperatura del refrigerante del vehículo

void	setTemperatura(String temperatura)	Define la temperatura del refrigerante del vehículo (De ser necesario)
String	getPosicion()	Retorna la posición del acelerador
void	setPosicion(String posicion)	Define la posición del acelerador (De ser necesario)
String	getCarga()	Retorna la carga del motor
void	setCarga(String carga)	Define la carga del motor (De ser necesario)
void	guardarParametro(Activity actividad)	Mediante un objeto HttpClientePar inserta los valores requeridos en la base de datos

Clase HttpCliente

La clase HttpCliente es la encargada de gestionar la comunicación con el servidor. En esta clase se especifica la dirección URL donde se encontrará el servicio desarrollado para almacenar registros en la base de datos correspondientes a los códigos de falla detectados, la clase se encarga además de manejar todos los hilos necesarios en la transmisión de información mediante internet e incluso implementa un mensaje que notifica al usuario que los datos se encuentran en proceso de envío. Los métodos implementados en la clase HttpCliente se muestran en la tabla 3.17.

Tabla. 3.17 Clase HttpCliente

Tipo	Método	Descripción
constructor	HttpCliente(Activity actividad)	Constructor de la clase
String	getMensajeCargaCabecera()	Obtener el String que se muestra en la cabecera del mensaje
void	setMensajeCargaCabecera(String mensaje)	Establecer el String que se muestra en la cabecera del mensaje
String	getMensajeCargaCuerpo()	Obtener el String que se muestra en el cuerpo del mensaje

void	setMensajeCargaCuerpo(String mensaje)	Establecer el String que se muestra en el cuerpo del mensaje
String	getServicioUrl()	Obtener la dirección URL del servidor
void	setServicioUrl(String ServicioUrl)	Establecer la dirección URL del servidor
String	getRespuestaCuerpo()	Obtener el String que aparecerá como respuesta
void	setRespuestaCuerpo(String RespuestaCuerpo)	Establecer el String que aparecerá como respuesta
boolean	estaInternetPermitido(Activity actividad)	True si la aplicación tiene los permisos para acceder a internet, caso contrario False
void	anadirValor(String nombre, String valor)	Inserta los valores en la base de datos
void	ejecutarHttpPost(String ServicioUrl)	Recibe la dirección URL
void	ejecutarHttpPostA(Activity actividad, String ServicioUrl)	Es el método que realiza la inserción de datos en el servicio php

Clase HttpClienteAuto

La clase HttpClienteAuto de manera similar es la encargada de gestionar la comunicación con el servidor. En esta clase se especifica la dirección URL donde se encontrará el servicio desarrollado para almacenar registros en la base de datos correspondientes a los automóviles diagnosticados, la clase se encarga además de manejar todos los hilos necesarios en la transmisión de información mediante internet e incluso implementa un mensaje que notifica al usuario que los datos se encuentran en proceso de envío. Los métodos implementados en la clase HttpClienteAuto se muestran en la tabla 3.18.

Tabla. 3.18 Clase HttpClienteAuto

Tipo	Método	Descripción
constructor	HttpClienteAuto(Activity actividad)	Constructor de la clase
String	getMensajeCargaCabecera()	Obtener el String que se muestra en la cabecera del mensaje
void	setMensajeCargaCabecera(String mensaje)	Establecer el String que se muestra en la cabecera del mensaje
String	getMensajeCargaCuerpo()	Obtener el String que se muestra en el cuerpo del mensaje
void	setMensajeCargaCuerpo(String mensaje)	Establecer el String que se muestra en el cuerpo del mensaje
String	getServicioUrl()	Obtener la dirección URL del servidor
void	setServicioUrl(String ServicioUrl)	Establecer la dirección URL del servidor
String	getRespuestaCuerpo()	Obtener el String que aparecerá como respuesta
void	setRespuestaCuerpo(String RespuestaCuerpo)	Establecer el String que aparecerá como respuesta
boolean	estaInternetPermitido(Activity actividad)	True si la aplicación tiene los permisos para acceder a internet, caso contrario False
void	anadirValor(String nombre, String valor)	Inserta los valores en la base de datos
void	ejecutarHttpPost(String ServicioUrl)	Recibe la dirección URL
void	ejecutarHttpPostA(Activity actividad, String ServicioUrl)	Es el método que realiza la inserción de datos en el servicio php

Clase HttpClientePar

La clase HttpClientePar es la encargada de gestionar la comunicación con el servidor en lo referente al monitoreo local. En esta clase se especifica la dirección URL donde se encontrará el servicio desarrollado para almacenar registros en la base de datos correspondientes a los parámetros de funcionamiento, la clase se encarga además de manejar todos los hilos necesarios en la transmisión de información mediante internet e implementa de forma similar a HttpCliente y HttpClienteAuto, un mensaje que notifica al usuario que los datos se encuentran en proceso de envío. Los métodos implementados en la clase HttpClientePar se muestran en la tabla 3.19.

Tabla. 3.19 Clase HttpClientePar

Tipo	Método	Descripción
constructor	HttpClientePar(Activity actividad)	Constructor de la clase
String	getMensajeCargaCabecera()	Obtener el String que se muestra en la cabecera del mensaje
void	setMensajeCargaCabecera(String mensaje)	Establecer el String que se muestra en la cabecera del mensaje
String	getMensajeCargaCuerpo()	Obtener el String que se muestra en el cuerpo del mensaje
void	setMensajeCargaCuerpo(String mensaje)	Establecer el String que se muestra en el cuerpo del mensaje
String	getServicioUrl()	Obtener la dirección URL del servidor
void	setServicioUrl(String ServicioUrl)	Establecer la dirección URL del servidor
String	getRespuestaCuerpo()	Obtener el String que aparecerá como respuesta
void	setRespuestaCuerpo(String RespuestaCuerpo)	Establecer el String que aparecerá como respuesta

boolean	estaInternetPermitido(Activity actividad)	True si la aplicación tiene los permisos para acceder a internet, caso contrario False
void	anadirValor(String nombre, String valor)	Inserta los valores en la base de datos
void	ejecutarHttpPost(String ServicioUrl)	Recibe la dirección URL
void	ejecutarHttpPostA(Activity actividad, String ServicioUrl)	Es el método que realiza la inserción de datos en el servicio php

3.4 Implementación del servidor

La aplicación móvil se encarga de la comunicación y la recolección de datos del automóvil. Sin embargo el proyecto abarca la creación de un servidor Web que permita alojar los códigos de falla pertenecientes a varios vehículos y observar registros de los parámetros recolectados en el monitoreo local.

3.4.1 Diseño de la base de datos

Para almacenar los códigos de falla registrados en un automóvil, se diseñó e implementó una base de datos que almacene información sobre los vehículos, los códigos de falla encontrados, el significado de cada código y los parámetros monitoreados.

La base de datos diseñada se denomina OBDII y está conformada por cuatro tablas: Auto, CódigoFalla, Error y Parámetro, relacionadas por los parámetros “Placa” y “Código”. En la figura 3.27 se presenta el modelo conceptual, mientras en la figura 3.28 se muestra el modelo físico de la base de datos.

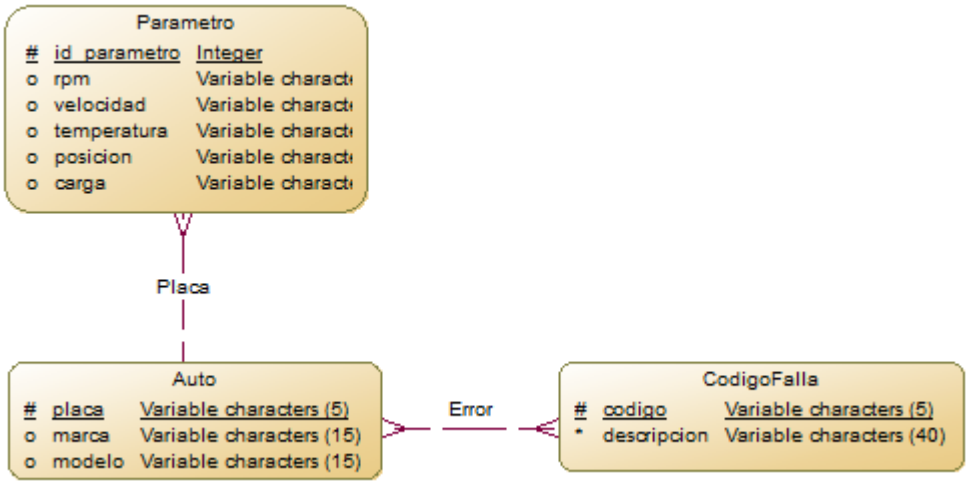


Figura. 3.27 Modelo conceptual de base de datos OBDII

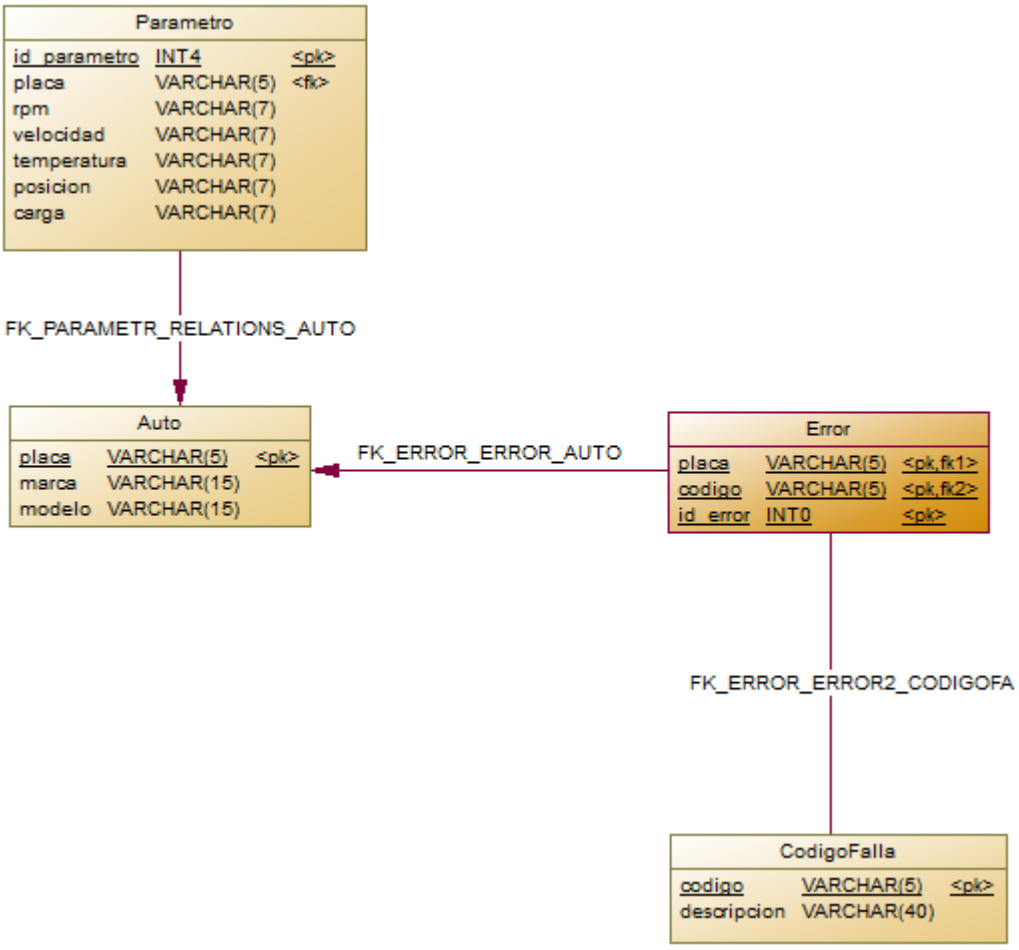


Figura. 3.28 Modelo físico de base de datos OBDII

Tabla Auto

La entidad Auto representa a los vehículos que serán monitoreados en el proyecto. Consta de tres campos. Placa, marca y modelo del automotor. El identificador primario y único de la tabla será la Placa del vehículo ya que no es posible tener dos vehículos con la misma placa. El registro de estos valores se realiza por el conductor del vehículo directamente en la aplicación móvil.

Tabla Parámetro

En la tabla parámetro se almacenan los valores correspondientes a la velocidad, rpm, temperatura del refrigerante, carga del motor y posición del acelerador del vehículo cada vez que el reloj del teléfono marca 0 segundos, es decir al comenzar un nuevo minuto.

Para identificar el automóvil al cual pertenece el conjunto de parámetros también se almacena en esta tabla un campo placa para que de esta forma pueda relacionarse con la tabla Auto. La clave primaria de la tabla es un identificador numérico que aumenta automáticamente al ingresarse un nuevo registro.

Tabla Error

La entidad Error es el eje principal de la base de datos. Cuando un vehículo envía los errores encontrados al servidor, estos se almacenan en la tabla Error. La entidad se compone de tres campos. Un identificador numérico único para cada error encontrado, el valor del código de falla correspondiente y la placa del vehículo donde fue hallado el error.

La clave primaria de la tabla es el identificador numérico, el cual se incrementa automáticamente cada vez que un nuevo registro es ingresado. Si bien en teoría la clave primaria debería ser el código de la falla encontrada, la necesidad de una clave primaria numérica se da debido a que varios autos pueden presentar el mismo problema y por tanto es necesario diferenciar claramente cada caso.

Tabla Código_Falla

Sin la tabla Código_Falla, el usuario podría observar los códigos de falla detectados en el automóvil pero no que significa cada uno. Código_Falla contiene almacenados todos los códigos de falla que se pueden obtener mediante OBDII junto con su significado. De esta forma el usuario tendrá una idea clara sobre donde se encuentra el problema del vehículo. Existen dos campos en la tabla. El código de la falla y la descripción correspondiente.

3.4.2 Desarrollo de aplicación Web

Interfaz de usuario

La interfaz de usuario del sitio se compone de cuatro ventanas. La primera ventana corresponde al inicio de sesión para el ingreso a la aplicación. La ventana de inicio de sesión se denomina index.php y su estructura se muestra en la figura 3.29, mientras su implementación se muestra en la figura 3.30. Los componentes fundamentales de esta ventana son dos campos de texto destinados al ingreso del nombre de usuario y contraseña y un botón que dirige la aplicación a la siguiente ventana si la autenticación es correcta.

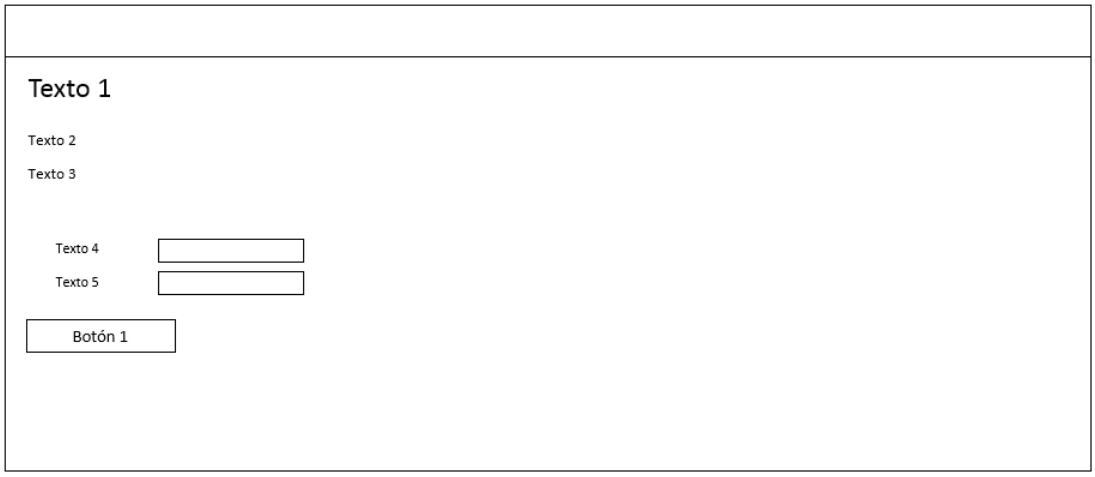


Figura. 3.29 Diseño ventana de inicio de sesión - aplicación web

Figura. 3.30 Ventana de inicio de sesión - aplicación web

La segunda ventana por su parte es la presentación de la aplicación web. El componente principal de esta ventana es un "combo box" que incluye las placas de los autos registrados en la base de datos y un botón que direcciona a la aplicación a una nueva ventana donde se muestra los datos específicos del vehículo al que corresponde la placa.

La ventana de presentación se implementó mediante un formulario denominado Index1.php en el cual se define la forma y estructura de la ventana. En la figura 3.31 se muestra la estructura de la ventana de acceso a la información mientras en la figura 3.32 se muestra la implementación de la ventana.

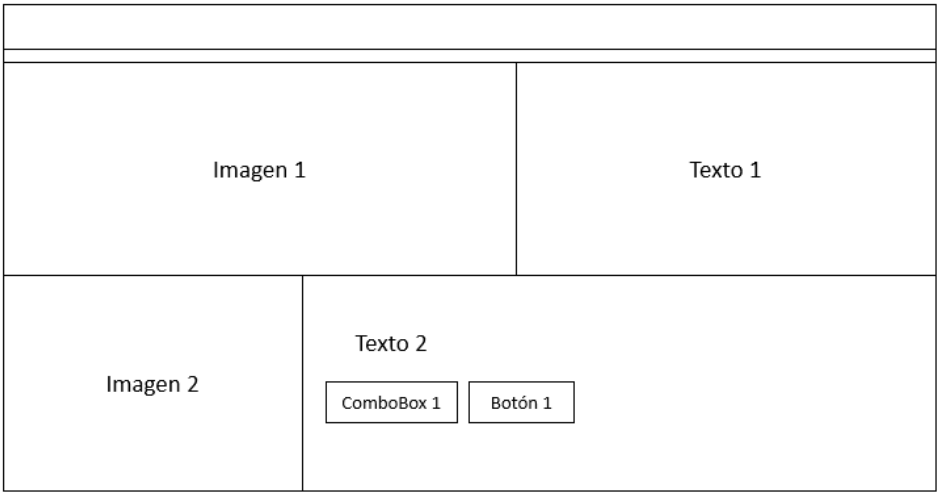


Figura. 3.31 Diseño ventana de acceso – aplicación web

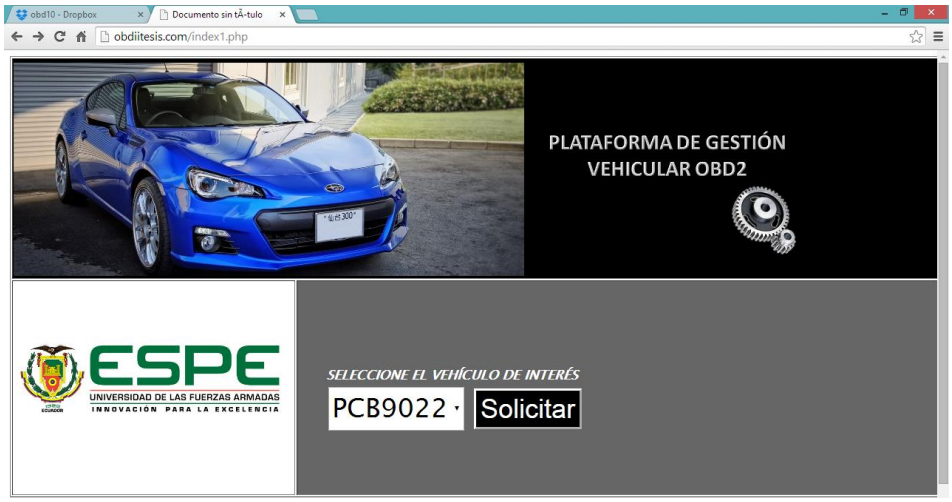


Figura. 3.32 Ventana de acceso - aplicación web

En la ventana a la que se direcciona la aplicación cuando se ha seleccionado un vehículo se muestra la información correspondiente al automóvil especificado.

Se presenta la placa, marca y modelo del automotor en el campo Texto 2, los códigos de todas las fallas encontradas en el vehículo durante el diagnóstico en el campo Lista 1 y en el campo Lista 2 el significado de dichos códigos. Además existe un botón que permite presentar al usuario, datos históricos de los parámetros monitoreados mediante el uso de gráficas. La implementación de esta ventana se realizó mediante un formulario denominado ventanaErrores.php. La figura 3.33 muestra el diseño de la ventana de presentación de información y la figura 3.34 muestra su implementación.

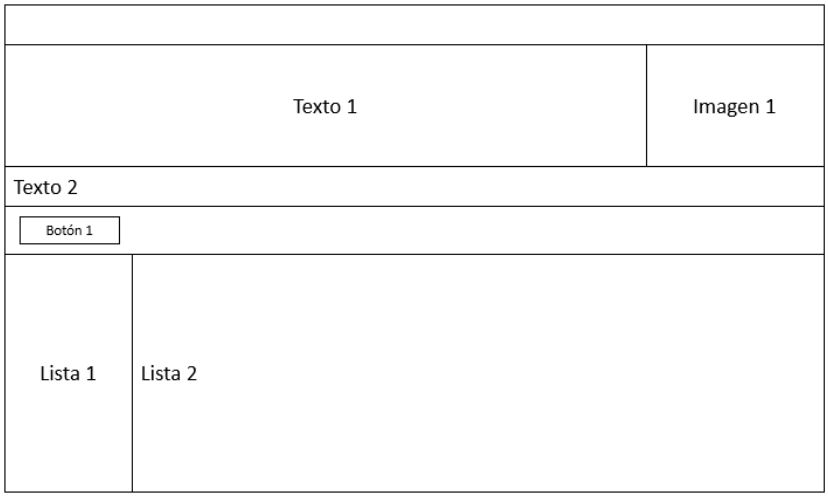


Figura. 3.33 Diseño ventana de información - aplicación web

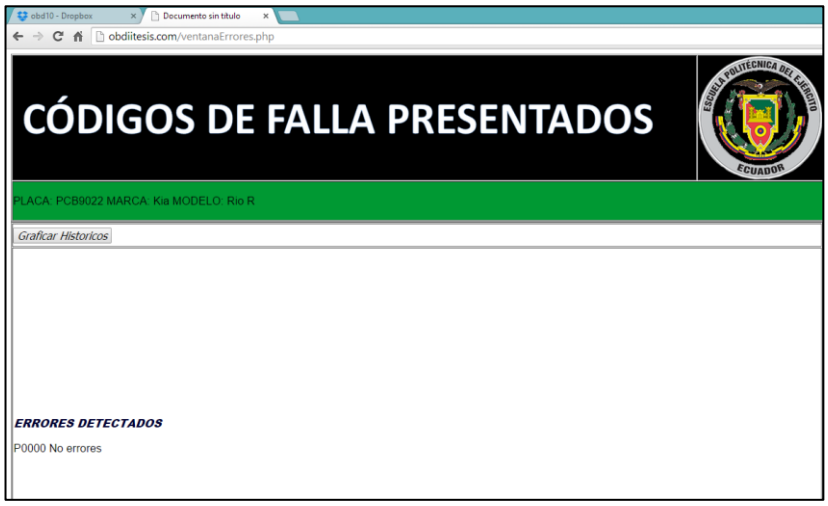


Figura. 3.34 Ventana de información - aplicación web

La ventana que presenta los datos recopilados en el monitoreo local se implementó mediante el formulario graficos.php. La ventana muestra cinco gráficas distintas con el mismo dominio pero diferente rango, cada una escalada de acuerdo al valor de la variable que representan. En la figura 3.35 se muestra la implementación de los gráficos de parámetros obtenidos.

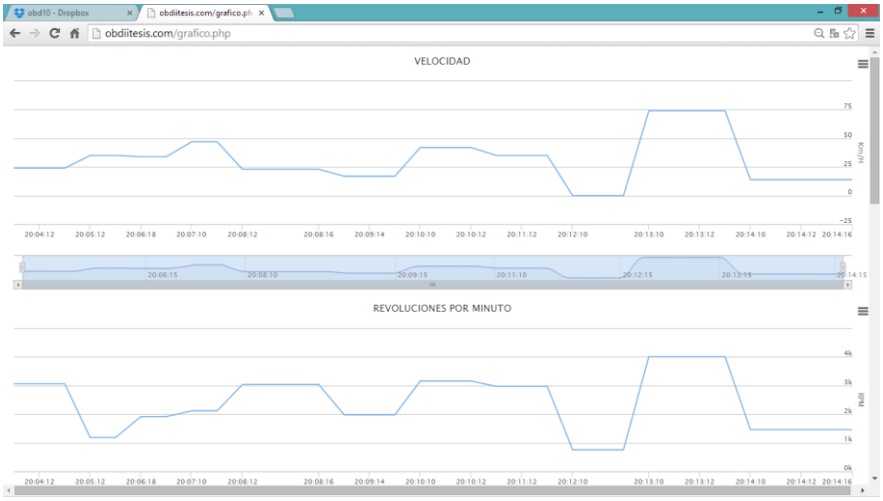


Figura. 3.35 Ventana de gráficas - aplicación web

Conexión con la base de datos

Tanto la aplicación móvil como la aplicación web interactúan entre sí mediante la base de datos diseñada para el proyecto. En este contexto, la aplicación web será la que administre directamente el ingreso de nuevos registros en las tablas de la base de datos, estos registros serán los que la aplicación móvil envíe mediante Internet.

Con el fin de almacenar la información de autenticación de la base de datos se creó la clase Globals.php. En esta clase se define el nombre o dirección del servidor donde se encuentra la base de datos, el nombre de la base de datos y finalmente el nombre de usuario y la contraseña de acceso.

En la tabla 3.20 se presentan los métodos implementados en Globals.php

Tabla. 3.20 Globals.php

Método	Descripción
getServidorBD()	Retorna el nombre o URL del servidor web
getNombreBD()	Retorna el nombre de la base de datos
getUsuarioBD()	Retorna el nombre de usuario para la autenticación
getClaveBD()	Retorna la contraseña para la autenticación

Para la realización de consultas se creó la clase gestorDB.php. Conjuntamente con Globals.php se encarga de establecer la conexión con la base de datos y mediante un método específico realiza una consulta específica a la base. Además se encarga de cerrar la conexión una vez realizada la consulta. El método implementado en la clase gestorBD se muestra en la tabla 3.21.

Tabla. 3.21 gestorBD.php

Método	Descripción
ejecutarConsulta($sql)	Recibe como argumento una sentencia sql correspondiente a la consulta o acción requerida. Retorna el resultado de la consulta

Inserción de registros en base de datos

Las clases descritas en la sección 3.4.3.2 se encargan de la conexión y consulta a la base de datos, Globals.php y gestorBD.php dan soporte a las clases encargadas de realizar como tal la actualización de datos en la base OBDII.

La inserción de registros en las tablas de la base de datos se realiza mediante las clases Errores.php, Parámetros.php y Autos.php. Las clases se encargan de añadir códigos de falla, parámetros de funcionamiento y automóviles en las tablas respectivas mediante sentencias sql. De ser necesario se puede utilizar estas clases para realizar pruebas de la aplicación sin los formularios de interfaz de usuario ya que también es posible visualizar en pantalla los valores registrados en las tablas de la base de datos OBDII. En la tabla 3.22 se muestran los métodos implementados en Errores.php mientras en la tabla 3.23 se muestran los métodos implementados en Autos.php y finalmente en la tabla 3.24 Parámetros.php.

Tabla. 3.22 Errores.php

Método	Descripción
getErrores()	Realiza una consulta sql a la base de datos en busca de los códigos de falla almacenados
getJSONError($tag, $errormsg)	Devuelve el resultado de una consulta en formato JSON para mostrarlo en pantalla
insertarError($codigo, $descripcion, $placa)	Inserta mediante una sentencia sql un nuevo registro en la tabla Error
getJSONInsertErrores($codigo, $descripcion, $placa)	Devuelve el resultado del ingreso del nuevo registro para mostrarlo en pantalla

Tabla. 3.23 Autos.php

Método	Descripción
getAutos()	Realiza una consulta sql a la base de datos en busca de los vehículos almacenados
getJSONAuto($tag, $errormsg)	Devuelve el resultado de una consulta en formato JSON para mostrarlo en pantalla
insertarAuto($placa, $marca, $modelo)	Inserta mediante una sentencia sql un nuevo registro en la tabla Auto
getJSONInsertAutos($placa, $marca, $modelo)	Devuelve el resultado del ingreso del nuevo registro en formato JSON para mostrarlo en pantalla

Tabla. 3.24 Parámetros.php

Método	Descripción
getParametros()	Realiza una consulta sql a la base de datos en busca de conjuntos de parámetros monitoreados
getJSONParametro($tag, $errormsg)	Devuelve el resultado de una consulta en formato JSON para mostrarlo en pantalla
insertarParametro($rpm, $velocidad, $temperatura, $posición, $carga, $placa)	Inserta mediante una sentencia sql un nuevo registro en la tabla Parámetro
getJSONInsertParámetro($rpm, $velocidad, $temperatura, $posición, $carga, $placa)	Devuelve el resultado del ingreso del nuevo registro en formato JSON para mostrarlo en pantalla

ServicioAnadirErrores.php por su parte es el formulario a donde apunta la aplicación móvil al añadir códigos de falla. Los valores a ingresar en la base de datos se envían a esta clase mediante el método POST para luego ser llamados los métodos de Errores.php y finalmente registrar los campos enviados por la aplicación móvil.

ServicioAnadirAutos.php por su parte actúa de forma muy similar con la única diferencia que este es el formulario a donde apunta la aplicación cuando requiere insertar un nuevo automóvil en la base de datos.

ServicioAnadirPar.php en cambio, es el formulario a donde apunta la aplicación móvil cuando realiza cada minuto el ingreso de parámetros para la elaboración de gráficas.

3.4.3 Alojamiento Web de la aplicación

Dado que PHP es un lenguaje que actúa en el lado del servidor, en el desarrollo de la aplicación web se utilizó el software XAMPP para convertir a un computador en un servidor web local. Sin embargo, existía la limitante que la aplicación móvil

únicamente podía enviar datos al servidor si tanto el computador como el teléfono se encontraban conectados a las misma red LAN.

Para solucionar este inconveniente se contrató el servicio de alojamiento web de la empresa Yamburara y se adquirió además el nombre de dominio www.obdiitesis.com.

Las características principales que oferta el servicio son:

- 2 GB de espacio de almacenamiento.
- 20 GB de transferencia.
- 1 dominio predeterminado.
- Capacidad de gestión del sitio mediante la herramienta cPanel.

Para realizar el proceso de migración de la aplicación web desde el servidor local hacia el sitio web es necesario actualizar la clase Globals.php con la información mostrada en la tabla 3.25.

Tabla. 3.25 Parámetros en Globals.php

Parámetro	Valor
Nombre del servidor web	localhost
Nombre de Base de Datos	obd2
Usuario	felipe
Clave	71mcl71

El nombre de servidor web tiene el valor de “localhost” debido a que los archivos PHP que conforman la aplicación y la base de datos estarán almacenados dentro del mismo directorio en el servidor de alojamiento.

Para poder completar la migración es necesario subir los archivos que componen la aplicación web mediante el protocolo FTP, esto se realizó utilizando el software para cliente FTP NicoFTP3**.**

Finalmente una vez cargados todos los archivos, se realizó un cambio en la dirección URL a la que apunta la aplicación móvil para enviar los códigos de falla. La nueva dirección colocada es la correspondiente al dominio contratado.

Una vez completada la migración es posible enviar los datos desde el teléfono al servidor sin necesidad de una red LAN.

CAPÍTULO 4

PRUEBAS DE FUNCIONAMIENTO DEL SISTEMA

4.1 Protocolo de pruebas

Considerando que el sistema desarrollado posee dos modos de operación, se determinaron de igual forma dos procedimientos de pruebas distintos.

4.1.1 Protocolo en modo de monitoreo local

En el modo de monitoreo local, el método escogido para probar el desempeño se basó en pruebas de campo con el sistema funcionado en el vehículo de prueba. Se realizaron recorridos de en promedio 20 minutos tanto en entornos urbanos como en carretera y se realizó filmaciones de los recorridos con el fin de detectar problemas en la lectura de tramas OBDII.

Se implementó además en la aplicación móvil, un campo de texto encargado de mostrar las tramas recibidas y enviadas por el teléfono antes de ser procesadas. Esta información permitió comprobar el formato de las tramas que están siendo procesadas e identificar conflictos en el cálculo e interpretación.

4.1.2 Protocolo en modo de diagnóstico

En el modo de diagnóstico se busca encontrar los códigos de falla almacenados en la computadora del vehículo, sin embargo si el automotor no presenta ningún problema, el sistema no detecta ningún código. El vehículo de prueba utilizado corresponde justamente a este caso por lo que no fue posible probar la lectura de códigos de errores con el mencionado vehículo.

Nació entonces la necesidad de desarrollar un mecanismo que permita realizar las pruebas del modo de diagnóstico, la solución definida fue el desarrollo de una

aplicación Java que simule a la computadora de un vehículo y proporcione los códigos de falla necesarios para la realización de las pruebas.

4.2 Monitoreo Local

El objetivo principal de las pruebas realizadas fue el constatar que los valores mostrados al conductor en la pantalla del teléfono, corresponden efectivamente a los valores reales de los parámetros medidos en el automóvil.

Gracias a los videos realizados se pudo analizar las tramas que en su debido momento necesitaron ajustes en su procesamiento y se alcanzó que la aplicación responda correctamente a los valores enviados por la computadora del vehículo.

Se realizaron 5 recorridos en ciudad y 5 recorridos en carretera donde se observó el rendimiento del sistema en cada parámetro del vehículo. La tabla 4.1 muestra la descripción de los 10 viajes realizados mientras los resultados se muestran en las tablas 4.2 y 4.3.

Tabla. 4.1 Descripción de recorridos

Ciudad			
Recorrido	**Duración (min)**	**Terreno**	**Hora de inicio**
Viaje 1	21	Asfalto	10:37
Viaje 2	21	Adoquín/Asfalto	11:10
Viaje 3	21	Asfalto	11:45
Viaje 4	23	Asfalto	12:14
Viaje 5	18	Adoquín/Asfalto	15:20
Carretera			
Recorrido	**Duración (min)**	**Terreno**	**Hora de inicio**
Viaje 1	23	Asfalto	06:00
Viaje 2	22	Asfalto	06:40
Viaje 3	16	Asfalto	07:15
Viaje 4	23	Asfalto	07:42
Viaje 5	16	Asfalto	08:35

Tabla. 4.2 Recorridos en ciudad

Parámetro	Viaje 1	Viaje 2	Viaje 3	Viaje 4	Viaje 5
RPM	✓	×	✓	✓	✓
Velocidad	✓	×	✓	✓	✓
Posición acelerador	✓	✓	✓	✓	✓
Temperatura refrigerante	✓	✓	✓	✓	✓
Carga del motor	✓	✓	✓	✓	✓

Tabla. 4.3 Recorridos Carretera

Parámetro	Viaje 1	Viaje 2	Viaje 3	Viaje 4	Viaje 5
RPM	✓	✓	✓	✓	✓
Velocidad	✓	✓	✓	✓	✓
Posición acelerador	✓	✓	✓	✓	✓
Temperatura refrigerante	✓	✓	✓	✓	✓
Carga del motor	✓	✓	✓	✓	✓

En los recorridos en carretera se pudo confirmar que los parámetros fueron leídos e interpretados correctamente. Ninguno de los valores mostró datos no coherentes en los cinco recorridos, sin embargo en las pruebas realizadas en ciudad durante el segundo viaje se observó que los parámetros “RPM” y “Velocidad” mostraban datos incorrectos en la pantalla. Se determinó que el dispositivo OBDII no se encontraba correctamente conectado al automóvil lo que provocó errores en la aplicación. Los parámetros “RPM” y “Velocidad” fueron los más afectados debido a que son los valores con mayor variación durante la conducción del automóvil.

En la figura 4.1 se observa al sistema en funcionamiento en el modo de monitoreo local.

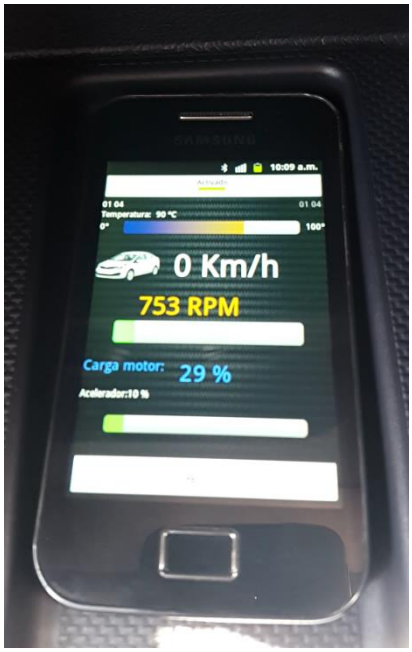


Figura. 4.1 Pruebas modo monitoreo local

4.3 Diagnóstico en vehículo de prueba

Como se mencionó en el inicio del presente capítulo, el vehículo de prueba utilizado para el desarrollo de la aplicación no contaba con errores almacenados en la computadora por lo que al someterlo al sistema desarrollado se arrojaron resultados mostrando que todo funciona correctamente.

Si bien no fue posible encontrar errores, el sistema demostró un correcto funcionamiento al mostrar que efectivamente el vehículo no almacenaba ningún código de falla y al enviar este dato al servidor los usuarios de la aplicación web pueden observar el mensaje “No Errores” indicando el buen estado del automotor. En la figura 4.2 se muestra el diagnóstico realizado al vehículo de prueba y además se muestra el resultado del mencionado diagnóstico visto desde la aplicación web.

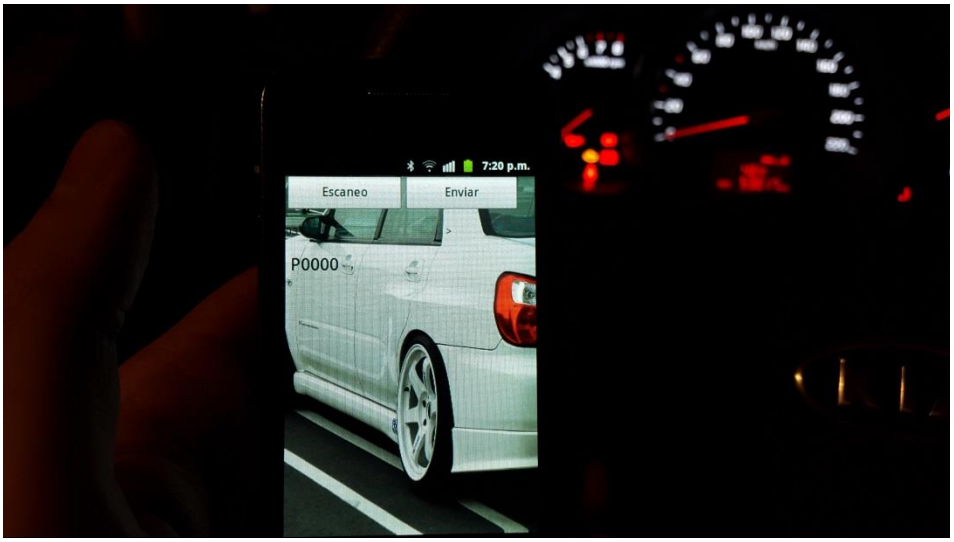


Figura. 4.2 Pruebas en modo diagnóstico

4.4 Simulador de códigos de falla

En vista de la indisponibilidad de un vehículo que tenga almacenados códigos de falla en su computadora, se desarrolló una aplicación Java que actúa como un simulador del automóvil a ser diagnosticado.

De igual forma que el dispositivo BAFX OBDII, el computador donde se ejecuta la aplicación desarrollada actúa como servidor en la comunicación Bluetooth, es decir responde a las solicitudes enviadas por la aplicación móvil.

El simulador recibe las mismas tramas que recibiría normalmente la computadora del automóvil. Cuando el comando solicitando el diagnóstico es recibido, la aplicación Java genera aleatoriamente códigos de falla definidos en el estándar OBDII. El número de códigos generados también se determina aleatoriamente, pudiendo generarse entre 1 a 10 códigos. El diagrama de flujo de la aplicación se presenta en la figura 4.3.

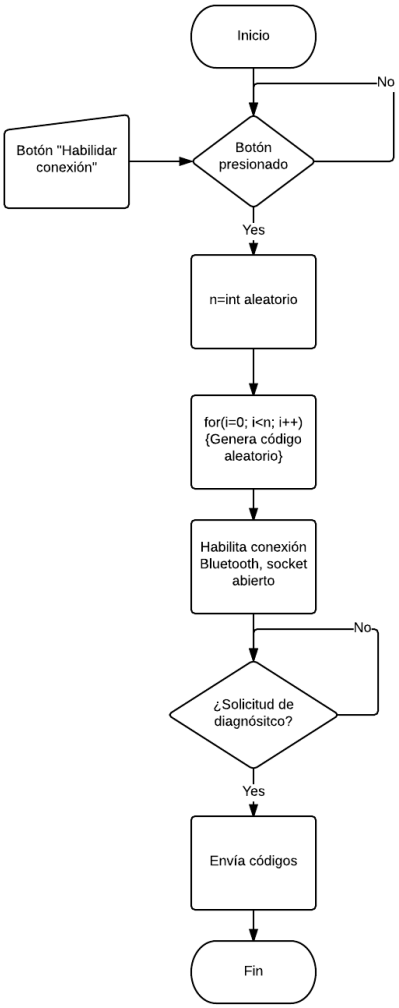


Figura. 4.3 Diagrama de flujo de simulador para pruebas

El formato de las tramas que son enviadas como respuesta desde el computador es el mismo que el utilizado por la computadora del vehículo. La aplicación desarrollada se compone de tres clases: La clase Simulador, Código y PuertoSerie.

4.4.1 Clase Simulador

La clase Simulador conforma la interfaz gráfica de la aplicación, dentro de esta clase se generan los objetos de tipo Código y PuertoSerie para gestionar el funcionamiento de la aplicación. Dentro de la interfaz de usuario se tiene dos componentes: Un campo de texto donde se presentan los códigos de falla generados y un botón encargado de generar los códigos y habilitar la comunicación Bluetooth.

4.4.2 Clase Código

La clase Código es la encargada de generar aleatoriamente los códigos de fallas que se enviarán al teléfono inteligente. La clase se compone de un método denominado GenerarCodigo, el mismo que retorna un arreglo de Strings donde cada posición del arreglo representa un código de falla.

4.4.3 Clase PuertoSerial

Cuando se crea un objeto de la clase PuertoSerial, se abre el socket para el establecimiento de la conexión Bluetooth entre el computador donde se ejecuta el simulador y la aplicación móvil. La clase únicamente posee un constructor que recibe como argumento un arreglo de Strings en la cual se encuentran los códigos de falla generados por la clase Código.

En la figura 4.4 se muestra la aplicación Java desarrollada y en la figura 4.5 su interacción con la aplicación móvil.

Figura. 4.4 Simulador generador de códigos de falla

Figura. 4.5 Interacción del simulador con la aplicación móvil

4.5 Funcionamiento de la aplicación móvil

La aplicación móvil desarrollada posee compatibilidad de funcionamiento con todas las versiones del sistema operativo Android desde la versión 2.3 en adelante. Se instaló la aplicación en varios terminales a fin de comprobar que son compatibles y la aplicación funciona correctamente. Las pruebas realizadas consistieron en:

- Establecer si la aplicación móvil no sufre de retrasos en la lectura de parámetros.
- Verificar que la interfaz gráfica no sufre distorsiones de acuerdo al tamaño de pantalla.
- Comprobar que el diagnóstico del vehículo se realiza correctamente.
- Determinar que la aplicación no sufre de congelamientos o interrupciones.

La tabla 4.4 muestra el resultado de las pruebas realizadas.

Tabla. 4.4 Compatibilidad de aplicación móvil

Versión	Terminal	Retardo	Interfaz gráfica	Diagnóstico	Consumo
2.3.3	Samsung ACE	✓	✓	✓	✓
4.4.2	Samsung S5	✓	✓	✓	✓
4.1.1	Huawei Y320	✓	✓	✓	✓

4.6 Rendimiento de la aplicación móvil

La aplicación móvil desarrollada requiere el consumo de varios recursos del teléfono inteligente para su funcionamiento. Los principales recursos que fueron evaluados son:

- Consumo de energía
- Uso de datos
- Memoria RAM

Partiendo con la batería del terminal al 100% de carga y un plan de datos de 15 MB activados, después de recorrer un trayecto de 20 Kilómetros de distancia en 35 minutos y con la aplicación en funcionamiento se obtuvo que el porcentaje de consumo de energía fue de 7% y el consumo de datos fue 460 KB. La memoria RAM consumida por la aplicación durante su ejecución según muestra el teléfono inteligente fue de 86 MB. En la tabla 4.5 se muestra un resumen del rendimiento de la aplicación móvil creada.

Tabla. 4.5 Rendimiento de aplicación móvil

Indicadores	Consumo en prueba	Consumo/hora
Datos	460 KB	789 KB/h
Memoria RAM	86 MB	86 MB
Batería	7%	12%/h

4.7 Funcionamiento de la aplicación web

Con el fin de probar el funcionamiento de la aplicación web se realizó el diagnóstico del vehículo de prueba y el diagnóstico mediante el simulador desarrollado. Los resultados en ambos casos se enviaron al servidor con el fin de constatar que los datos no se perdieron. Las cuatro pruebas realizadas consistieron en:

- Prueba 1: Se utilizó la aplicación Java desarrollada para simular la generación de fallas en un vehículo. El simulador generó aleatoriamente ocho códigos.
- Prueba 2: De igual forma se repitió la prueba con el simulador, en esta ocasión se generó un solo código
- Prueba 3: Se realizó el diagnóstico sobre el vehículo de prueba, arrojando como resultado un código. El código arrojado P0000 indica la ausencia de errores en el vehículo.
- Prueba 4: Se utilizó el software de terminal Tera Term para enviar tres tramas simulando códigos de falla.

Una vez recibidos los códigos de falla, los mismos fueron enviados al servidor, los resultados de estas pruebas se muestran en la tabla 4.6.

Tabla. 4.6 Pruebas de aplicación web

Parámetros	Prueba 1	Prueba 2	Prueba 3	Prueba 4
Códigos de falla enviados	8	1	1	3
Códigos de falla recibidos	8	1	1	3
Porcentaje de éxito	100%	100%	100%	100%

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

El sistema desarrollado permite visualizar las revoluciones por minuto del motor, la velocidad del vehículo, la temperatura del refrigerante, el porcentaje de carga del motor y la posición del acelerador de un vehículo en la pantalla de un teléfono Android.

La aplicación móvil implementada realiza un diagnóstico del estado mecánico del vehículo mediante el protocolo OBDII. Se obtiene como resultado códigos de falla que son almacenados en un servidor web, mediante un computador con conexión a internet es posible acceder a los códigos de falla extraídos del automóvil y conocer el significado.

La implementación del algoritmo de lectura de datos demostró que las tramas extraídas del automotor toman varios formatos distintos del teóricamente esperado, la principal causa de este fenómeno es la mayor capacidad de procesamiento que posee el terminal móvil, por este motivo la conformación de varias expresiones regulares que abarquen todos los formatos posibles es indispensable.

La lectura de las revoluciones por minuto del motor requirió especial atención debido a la estructura propia de la trama que representa a este parámetro. A diferencia de los tres bytes que componen las tramas de la velocidad, carga de motor, temperatura de refrigerante y posición del acelerador, la trama perteneciente a las revoluciones por minuto está conformada por cuatro bytes. Esto debido principalmente a que el parámetro está cuantificado en el orden de miles a diferencia de los demás parámetros cuantificados en el orden de los cientos.

En un principio se detectó que la lectura de parámetros en el monitoreo local era incorrecta debido a que la aplicación móvil enviaba nuevas solicitudes cuando el automóvil aún no había respondido a la solicitud anterior. Por esto fue necesario dar una pausa de 20 milisegundos a la aplicación para que el automóvil tenga el tiempo suficiente para responder las solicitudes.

Al momento de realizar el diagnóstico los códigos de falla encontrados se presentan en un determinado orden. Al enviar los datos al servidor, la aplicación web los muestra en un orden diferente, sin embargo esto no repercute en ningún aspecto el resultado del diagnóstico.

La aplicación Web desarrollada permite a un usuario del sistema acceder a través de internet a todos los códigos de falla detectados en un vehículo con su respectivo significado.

Gracias a la aplicación Web, el usuario es capaz de realizar un seguimiento de la velocidad, revoluciones por minuto, temperatura del refrigerante, carga del motor y posición del acelerador mediante gráficos históricos del comportamiento de estos parámetros.

Con el fin de comprobar el correcto funcionamiento de la aplicación móvil al diagnosticar un vehículo, se desarrolló una aplicación Java que actúa como un simulador de automóviles, el cual permitió verificar la óptima lectura de códigos de falla por parte del sistema.

El algoritmo de interpretación de los datos enviados desde el vehículo al terminal móvil se diseñó mediante el uso de expresiones regulares. Las expresiones regulares permiten que la aplicación sea capaz de discriminar correctamente las tramas enviadas desde el automóvil evitando la presentación de información errónea.

5.2 Recomendaciones

Se recomienda que al utilizar la aplicación móvil se evite el emparejamiento Bluetooth entre el teléfono y el equipo de audio propio del vehículo dado que el terminal móvil es incapaz de conectarse simultáneamente con dos dispositivos Bluetooth.

El diagnóstico del automotor puede realizarse con el motor encendido o con el motor apagado y la llave en la posición “ON”, sin embargo se recomienda realizar el diagnóstico con el motor apagado dado que los códigos de errores se encuentran almacenados en el computador del vehículo y no es necesario que el motor esté en funcionamiento.

Los datos correspondientes a la lectura de parámetros son enviados por el vehículo únicamente cuando el motor está encendido. Bajo estas circunstancias es recomendable que se active el monitoreo local del automóvil cuando el motor ya esté funcionando y la información pueda ser enviada.

El diagnóstico permite a los usuarios del sistema conocer si el vehículo presenta fallas y el significado de dichas fallas, la cuales están definidas en el estándar OBDII, sin embargo todo resultado obtenido debe ser validado por un especialista automotriz antes de realizar cualquier acción.

Se recomienda como trabajo futuro implementar animaciones en la interfaz gráfica de la aplicación móvil, esto facilitaría la visualización de la información y mejoraría la experiencia de usuario.

El sistema tiene el potencial para en un futuro apagar el indicador de mal funcionamiento o indicador de “check engine” mediante la aplicación móvil utilizando los comandos establecidos para dicha función en el estándar OBDII.

El servidor web actualmente es un actor de monitoreo del sistema. En un futuro se puede otorgar al servidor y a la aplicación web la capacidad para controlar el diagnóstico y la lectura de los parámetros del vehículo.

Se recomienda para futuros proyectos extender el monitoreo de parámetros del vehículo más allá de los cinco escogidos para el presente sistema. Los parámetros a monitorear pueden escogerse en base al enfoque que se requiera dar al sistema.

BIBLIOGRAFÍA

[1] J. Van Gilder, On-board diagnostics (OBD) history and current status, Ortega, 2005.

[2] NAPA Institute of Automotive Technology, OBD II and Second Generation Scan Tools, NAPA Institute of Automotive Technology, 1998.

[3] J. M. C. Ortega, Smartphone: toda la informacion al alcance de tu mano, Telos: Cuadernos de comunicación e innovación, 2010.

[4] Android. [En línea]. Available: http://www.android.com/phones-and-tablets/. [Último acceso: 5 Agosto 2014].

[5] Organización Mundial de la Salud, Informe sobre la situación mundial de la seguridad vial 2013, OMS, 2013.

[6] Agencia Nacional de Transito, «Siniestros Enero-Junio 2014,» Quito, 2014.

[7] «BAFX Products,» [En línea]. Available: http://www.bafxproducts.com/BAFX-Products-Bluetooth-diagnostics-Android. [Último acceso: 6 Agosto 2014].

[8] OBD Experts, « OBD Protocol. FAQ,» 2014. [En línea]. Available: http://www.obdexperts.co.uk. [Último acceso: 28 Septiembre 2014].

[9] NYSDEC, «Onboard Diagnostics. MIL,» [En línea]. Available: http://www.dec.ny.gov/chemical/8621.html. [Último acceso: 30 Septiembre 2014].

[10] J. Haartsen, «The Bluetooth radio system,» *Personal Communications IEEE 7*, nº 1, pp. 28-36, 2000.

[11] IEEE, «IEEE 802.15 WPAN Task Group 1,» [En línea]. [Último acceso: 12 Octubre 2014].

[12] F. M. A. Papacetzi, «Wireless Personal Area Network (WPAN) a Home Networking,» 2003. [En línea]. [Último acceso: 21 Octubre 2014].

[13] Android Open Source Project, «Philosophy and Goals,» 2011. [En línea]. [Último acceso: 1 Octubre 2014].

[14] Android Developers, «Bluetooth,» [En línea]. [Último acceso: 19 Diciembre 2014].

[15] J. E. Friedl, Mastering regular expressions, O'Reilly Media, Inc., 2002.

[16] C. J. Date y S. L. M. R. Faudón, Introducción a los sistemas de bases de datos, Pearson Educación, 2001.

[17] R. Ramakrishnan, J. Gehrke y J. Gehrke, Database management systems (Vol. 3), New York: McGraw-Hill, 2003.

[18] G. M. Nijssen y T. A. Halpin, Conceptual Schema and Relational Database Design: a fact oriented approach, Prentice-Hall, Inc., 1989.

[19] MySQL, «About SQL,» [En línea]. Available: http://www.mysql.com/about/. [Último acceso: 22 Diciembre 2014].

[20] L. Welling y L. Thomson, PHP and MySQL Web development, Sams Publishing, 2003.

[21] J. Adell y C. Bellver, «La internet como teleraña: el World-Wide Web.,» *Métodos de Información,* pp. 25-32., 2010.