



UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

ÁREA TÉCNICA

**TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y
COMPUTACIÓN**

**Extracción de datos de los sensores de un vehículo y geo-localización
mediante una aplicación móvil Android.**

TRABAJO DE TITULACIÓN.

AUTOR: Morocho Yunga, Diego Patricio

DIRECTOR: Riofrío Calderón, Guido Eduardo, Mg.

LOJA-ECUADOR

2016



Esta versión digital, ha sido acreditada bajo la licencia Creative Commons 4.0, CC BY-NY-SA: Reconocimiento-No comercial-Compartir igual; la cual permite copiar, distribuir y comunicar públicamente la obra, mientras se reconozca la autoría original, no se utilice con fines comerciales y se permiten obras derivadas, siempre que mantenga la misma licencia al ser divulgada. <http://creativecommons.org/licenses/by-nc-sa/4.0/deed.es>

Septiembre, 2016

APROBACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN

Mg.

Guido Eduardo Riofrío Calderón

DOCENTE DE LA TITULACIÓN

De mi consideración:

El presente trabajo de titulación: “Extracción de datos de los sensores de un vehículo y geo-localización mediante una aplicación móvil Android” realizado por Morocho Yunga Diego Patricio, ha sido orientado y revisado durante su ejecución, por cuanto se aprueba la presentación del mismo.

Loja, julio de 2016

f).....

Mg. Guido Eduardo Riofrío Calderón

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS

“Yo, Morocho Yunga Diego Patricio declaro ser autor (a) del presente trabajo de titulación: Extracción de datos de los sensores de un vehículo y geo-localización mediante una aplicación móvil Android, de la Titulación Sistemas Informáticos y Computación, siendo Guido Eduardo Riofrío Calderón director (a) del presente trabajo; y eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales. Además certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo, son de mi exclusiva responsabilidad.

Adicionalmente, declaro conocer y aceptar la disposición del Art. 88 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: “Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado o trabajos de titulación que se realicen con el apoyo financiero, académico o institucional (operativo) de la Universidad”.

f.....

Diego Patricio Morocho Yunga

1104250319

DEDICATORIA

Dedico este trabajo a Dios, por haberme dado la fortaleza de espíritu para superarme y alcanzar mis metas.

A mis padres Mauro y Mercedes, que me brindaron la oportunidad de alcanzar esta meta, son mi fuerza y mi razón para seguir adelante.

A Juan Carlos y Audrey Elizabeth, quienes con su ayuda, consejo y guías he podido terminar con éxito esta etapa de mi vida académica.

A mis pequeños sobrinos, que son la inspiración de muchas de mis metas.

AGRADECIMIENTO

Como todo proyecto de vida, siempre está lleno de momentos difíciles en los cuales se necesita de alguien, ya sea familia o amigos.

Agradezco a mi familia por haberme ayudado durante tanto tiempo, principalmente por su paciencia y consejos para desempeñarme en mi vida académica. Cada logro conseguido, es de ellos también.

A mis amigos y amigas, que durante la vida estudiantil se unieron a mí no solo como compañeros de clases, también como compañeros de experiencias, de esas que se vuelven anécdotas con el pasar de los años.

Al Mg. Guido Riofrío, director de mi trabajo de fin de titulación, gracias a su confianza, sus guías y experiencia pude realizar con éxito el trabajo encomendado.

A los ingenieros Jorge Cordero y Rodrigo Barba, quienes con sus consejos y enseñanzas me ayudaron a realizar de la mejor manera el trabajo de fin de titulación.

A cada docente que fue mi tutor en clases, por transmitir su experiencia y sus enseñanzas conmigo.

ÍNDICE DE CONTENIDOS

CARATULA.....	i
APROBACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN.....	ii
DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS.....	iii
DEDICATORIA	iv
AGRADECIMIENTO	v
ÍNDICE DE CONTENIDOS.....	vi
ÍNDICE DE IMÁGENES	xi
ÍNDICE DE TABLAS	xiv
RESUMEN.....	1
ABSTRACT	2
INTRODUCCIÓN.....	3
CAPÍTULO 1: MARCO TEÓRICO	4
1.1. Introducción.....	5
1.1.1. Historia de los sistemas de diagnóstico a bordo.....	5
1.1.2. Contaminación Vehicular	5
1.2. Sistemas de diagnóstico a bordo: OBD	6
1.3. Sistemas de diagnóstico a bordo versión II: OBDII.....	6
1.4. Aplicaciones móviles.....	7
1.4.1. Sistemas operativos móviles	7
1.5. Android OS	8
1.5.1. Arquitectura de aplicaciones Android.....	8
1.5.2. Aplicaciones Android.....	9
1.6. Aplicaciones Web.....	9
1.6.1. Desarrollo de aplicaciones web.....	10
1.6.1.1. Arquitectura cliente-servidor	10
1.6.2. Tecnologías de desarrollo.....	10
1.6.2.1. Lenguaje HTML	10
1.6.2.2. Estilos CSS.....	11
1.6.2.3. JavaScript	11
1.6.2.4. PHP.....	11
1.6.2.5. Servidor de Datos MySQL	11
CAPÍTULO 2: DISEÑO DE LA SOLUCIÓN	12

2.1.	Introducción	13
2.2.	Solución Propuesta	13
2.2.1.	Objetivo General.....	13
2.2.2.	Objetivos Específicos.....	13
2.3.	Descripción de la Solución.....	13
2.3.1.	Hardware OBD-II	15
2.3.1.1.	Selección del scanner OBDII.....	15
2.3.1.1.1.	Prueba 1.	15
2.3.1.1.2.	Resultados.....	17
2.3.1.1.3.	Prueba 2.	19
2.3.1.1.4.	Resultados.....	19
2.3.2.	Aplicación Móvil Android	20
2.3.2.1.	Arquitectura de la aplicación.....	20
2.3.2.2.	Módulo de Lectura OBD-II	20
2.3.2.3.	Módulo de Geo-localización.....	21
2.3.2.4.	Módulo de Gestión de datos.....	22
2.3.2.5.	Módulo de Configuración	23
2.3.2.6.	Servidor de Datos.....	24
2.3.3.	Aplicación Web	25
2.3.3.1.	Módulo de Carga de datos.....	26
2.3.3.2.	Módulo de Visualización de datos	27
2.3.3.3.	Módulo de Exportación de datos.....	28
2.4.	Alcance de la solución	28
2.4.1.	Aplicación Móvil Android	28
2.4.2.	Aplicación Web	29
2.4.3.	Vehículos	29
2.4.4.	Hardware OBDII.....	29
2.4.5.	Otras consideraciones	29
2.5.	Metodología de Desarrollo	29
2.5.1.	Extreme Programming XP	29
2.5.1.1.	Fases de XP	30
2.5.2.	Plan de Entregas	30
2.6.	Herramientas de desarrollo.....	31
2.6.1.	Hosting, Dominio: Heroku	32

2.6.2. Servidor de Base de Datos: ClearDB.....	32
CAPÍTULO 3: DESARROLLO.....	33
3.1. Proyecto en Eclipse.....	34
3.2. Módulo de geo-localización	35
3.2.1. Iteración 1.....	35
3.2.1.1. Clase geo.java	35
3.2.2. Iteración 2.....	36
3.3. Módulo de lectura OBD-II.....	37
3.3.1. Iteración 1.....	38
3.3.1.1. Clase bluetooth.java.....	38
3.3.1.2. Clase obd.java	39
3.3.2. Iteración 2.....	40
3.3.2.1. Clase obd.java	40
3.4. Módulo de gestión de datos.....	42
3.4.1. Iteración 1.....	42
3.4.1.1. Clase testConectividad.java.....	42
3.4.2. Iteración 2.....	43
3.4.2.1. Clase gestionDatos.java.....	43
3.4.2.1.1. Almacenamiento en el dispositivo.....	43
3.4.2.1.2. Transmisión al servidor.....	44
3.5. Módulo de configuración	45
3.5.1. Iteración 1.....	45
3.5.1.1. Clase leerConfig.java.....	45
3.6. Módulo web de carga de datos	48
3.6.1. Iteración 1.....	48
3.6.2. Iteración 2.....	49
3.6.2.1. Interfaz web de subida de archivo CSV	49
3.6.2.2. Subida de datos desde la aplicación.....	52
3.7. Módulo web de visualización de datos.....	54
3.7.1. Iteración 1.....	54
3.7.1.1. Interfaz de consulta.....	54
3.7.1.2. Interfaz de Visualización.....	55
3.7.1.3. Consumo del API Google Maps	56
3.7.2. Iteración 2.....	58

3.8.	Módulo web de exportación de datos.....	59
3.8.1.	Iteración 1.....	59
3.8.1.1.	Exportación en formato CSV	60
3.8.1.2.	Exportación en formato JSON.....	61
CAPÍTULO 4: PRUEBAS E IMPLEMENTACIÓN		63
4.1.	Elementos de prueba.....	64
4.1.1.	Vehículo de prueba	64
4.1.2.	Dispositivo de prueba	64
4.1.3.	Área de prueba	65
4.2.	Pruebas de Validación.....	65
4.2.1.	Prueba 1: Recorrido Local	65
4.2.1.1.	Aplicación Móvil	66
4.2.1.2.	Aplicación Web	69
4.2.2.	Prueba 2: Recorrido Inter-cantonal.....	70
4.2.2.1.	Aplicación Móvil	70
4.2.2.2.	Aplicación Web	73
4.2.3.	Prueba 3: Condiciones especiales.....	74
1.1.	Pruebas de Aceptación.....	74
1.1.1.	Aplicación Móvil	74
1.1.2.	Aplicación Web	76
1.2.	Pruebas de Rendimiento.....	78
1.2.1.	Aplicación Móvil	78
1.2.1.1.	Consumo de datos móviles.....	78
1.2.1.2.	Espacio de almacenamiento interno	79
1.2.1.3.	Uso de memoria RAM.....	80
1.2.2.	Aplicación Web	81
1.2.2.1.	Velocidad de carga.....	81
1.2.2.2.	Validación de etiquetas HTML.....	81
1.2.2.3.	Validación de estilos CSS	82
CONCLUSIONES.....		83
RECOMENDACIONES.....		84
TRABAJOS FUTUROS		85
BIBLIOGRAFÍA		86
ANEXOS		88

Anexo 1. Consumo del API Google Maps.....	89
Anexo 2. Hosting de aplicaciones en la plataforma de Heroku.	92
Hosting en Heroku.....	92
Servidor de Base de Datos: ClearDB.....	93
Anexo 3. Hardware y circuitos OBD-II.	95

ÍNDICE DE IMÁGENES

Figura 1. Elementos contaminantes de un vehículo.....	5
Figura 2. Entrada y pines OBDII	6
Figura 3. Arquitectura de Android OS.....	8
Figura 4. Flujo de la solución propuesta.....	14
Figura 5. Esquema de la solución.	15
Figura 6. Scanner OBDII Wifi-USB.....	16
Figura 7. Samsung Galaxy Tab 3.....	16
Figura 8. Aplicación Torque.....	17
Figura 9. Pruebas del scanner WiFi-USB en modo WiFi.....	17
Figura 10. Pruebas del scanner WiFi-USB en modo USB.	18
Figura 11. Pruebas con el scanner WiFi-USB con iPad.	18
Figura 12. Scanner OBDII Bluetooth.....	19
Figura 13. Pruebas del scanner Bluetooth en la Tablet Tab 3.....	19
Figura 14. Arquitectura de la aplicación móvil.	20
Figura 15. Diagrama de flujo del módulo de lectura OBD-II.....	21
Figura 16. Diagrama de flujo del módulo de geo-localización.	22
Figura 17. Diagrama de flujo del módulo de gestión de datos.....	23
Figura 18. Diagrama de flujo del módulo de configuración.	24
Figura 19. Estructura de la base de datos.	25
Figura 20. Arquitectura de la aplicación web.....	26
Figura 21. Diagrama de flujo del módulo web de carga de datos.	26
Figura 22. Diagrama de flujo del módulo web de visualización de datos.....	27
Figura 23. Diagrama de flujo del módulo web de exportación de datos.....	28
Figura 24. Fases de la metodología XP.	30
Figura 25. Creación de un proyecto de Android en Eclipse IDE.	34
Figura 26. Estructura de archivos del proyecto Android.	35
Figura 27. Módulo de Geo-localización - declaración de parámetros.....	35
Figura 28. Módulo de Geo-localización - Estado de red y GPS.	36
Figura 29. Módulo de Geo-localización - Ubicación vía red.	36
Figura 30. Módulo de Geo-localización - Ubicación vía GPS.	36
Figura 31. Módulo de Geo-localización - Obtener coordenadas vía red.....	37
Figura 32. Módulo de Geo-localización - Obtener coordenadas vía GPS.	37
Figura 33. Módulo de Lectura OBDII - Interfaz Bluetooth y búsqueda.	38
Figura 34. Módulo de Lectura OBDII - Conexión Bluetooth con un dispositivo.	39
Figura 35. Módulo de Lectura OBDII - Comandos OBDII.....	40
Figura 36. Módulo de Lectura OBDII – Métodos de envío de comandos OBDII.	40
Figura 37. Módulo de Lectura OBDII - Envío secuencial de comandos OBDII.....	41
Figura 38. Módulo de Lectura OBDII - Recepción de datos.....	42
Figura 39. Módulo de Gestión de Datos - Clase para determinar la conectividad.....	43
Figura 40. Módulo de Gestión de Datos - Ruta de almacenamiento de datos.....	43
Figura 41. Módulo de Gestión de Datos - Creación de archivos en el dispositivo.....	43
Figura 42. Módulo de Gestión de Datos - Inserción de datos en el CSV.....	44
Figura 43. Módulo de Gestión de Datos - Subida de datos al servidor.	44
Figura 44. Módulo de Configuración - Estructura del archivo de configuración.....	45

Figura 45. Módulo de configuración - Lectura del archivo de configuración.....	46
Figura 46. Módulo de Configuración - Obtener parámetro de tiempo.	46
Figura 47. Módulo de Configuración - Obtener parámetro identificador.	47
Figura 48. Módulo de Configuración - Obtener parámetro servidor.....	47
Figura 49. Interfaz de la aplicación móvil.	48
Figura 50. Módulo Web de Carga de Datos - Base de datos.....	48
Figura 51. Módulo Web de Carga de Datos - Script de base de datos.	49
Figura 52. Módulo Web de Carga de Datos - Formulario de subida de archivo CSV...	49
Figura 53. Módulo Web de Carga de Datos - Interfaz de subida de archivo CSV.....	50
Figura 54. Módulo Web de Carga de Datos - Lectura de archivo CSV.	50
Figura 55. Módulo Web de Carga de Datos - Conversión de datos hexadecimales.....	51
Figura 56. Módulo Web de Carga de Datos - Inserción de datos.....	52
Figura 57. Módulo Web de Carga de Datos - Conversión de hexadecimal desde la aplicación móvil.	53
Figura 58. Módulo Web de Carga de Datos - Inserción de datos enviados desde la aplicación móvil.	53
Figura 59. Módulo Web de Visualización - Declaración de librerías.....	54
Figura 60. Módulo Web de Visualización - Formulario de consulta.	54
Figura 61. Módulo Web de Visualización - Interfaz de consulta.	55
Figura 62. Módulo Web de Visualización – Declaración de API Google Maps.....	55
Figura 63. Módulo Web de Visualización – Interfaz de visualización.	56
Figura 64. Módulo Web de Visualización - Inicialización del mapa de Google.	56
Figura 65. Módulo Web de Visualización - Interfaz de visualización	57
Figura 66. Módulo Web de Visualización - Consulta de datos.....	57
Figura 67. Módulo Web de Visualización – Impresión de ubicaciones en el mapa.....	58
Figura 68. Módulo Web de Visualización de Datos - Trazar rutas entre ubicaciones...	59
Figura 69. Módulo Web de Exportación de Datos - Interfaz de descarga.....	60
Figura 70. Módulo Web de Exportación de Datos - Consulta de datos.....	60
Figura 71. Módulo Web de Exportación de Datos - Exportación de archivo CSV.	61
Figura 72. Módulo Web de Exportación de Datos - Exportación de archivo JSON.....	61
Figura 73. Módulo Web de Exportación de Datos - Interfaz de exportación de archivos.	62
Figura 74. Entrada OBDII del auto Toyota Rav4 2015.....	64
Figura 75. Dispositivo de pruebas Samsung Galaxy Note 4.....	65
Figura 76. Prueba de validación 1 – Recorrido local: Punto de inicio.....	66
Figura 77. Prueba de validación 1 - Recorrido local: Ejecución de la aplicación.....	67
Figura 78. Prueba de validación 1 – Recorrido local: Datos enviados de la aplicación.	67
Figura 79. Prueba de validación 1 - Recorrido local: Ejecución en segundo plano.....	68
Figura 80. Prueba de validación 1 - Recorrido local: Punto de llegada.....	69
Figura 81. Prueba de validación 1 - Recorrido local: Resultados en la aplicación web.	70
Figura 82. Prueba de validación 2 - Recorrido inter-cantonal: Punto de inicio.....	71
Figura 83. Prueba de validación 2 - Recorrido inter-cantonal: Punto de llegada.....	72
Figura 84. Prueba de validación 2 - Recorrido inter-cantonal: Datos almacenados en CSV.	72

Figura 85. Prueba de validación 2 - Recorrido inter-cantonal: Importación del archivo CSV.	73
Figura 86. Prueba de validación 2 - Recorrido inter-cantonal: Resultados en la aplicación web.	73
Figura 87. Prueba bajo condiciones especiales.	74
Figura 88. Pruebas de rendimiento - Aplicación móvil: Consumo de datos móviles.	79
Figura 89. Pruebas de rendimiento - Aplicación móvil: Tamaño de la aplicación.	80
Figura 90. Pruebas de rendimiento - Aplicación móvil: Consumo de memoria RAM.	80
Figura 91. Pruebas de rendimiento - Aplicación web: Velocidad de carga.	81
Figura 92. Pruebas de rendimiento - Aplicación web: Validación de código HTML.	82
Figura 93. Pruebas de rendimiento - Aplicación web: Validación de estilos CSS.	82
Figura 94. Consola de Desarrollo de Google.	89
Figura 95. Descripción del API JavaScript Google Maps.	90
Figura 96. Obtención de la clave de acceso al API Google Maps.	90
Figura 97. Creando la clave de acceso al API Google Maps.	90
Figura 98. Clave de Navegador API Google Maps.	91
Figura 99. Autenticación en Heroku Toolbelt.	92
Figura 100. Creación de una aplicación en Heroku.	93

ÍNDICE DE TABLAS

Tabla 1. Modos de trabajo del estándar OBDII.	7
Tabla 2. Plan de Iteraciones	31
Tabla 3. Herramientas de Desarrollo.	31
Tabla 4. Comandos OBDII a usar.....	39
Tabla 5. Áreas de prueba de la aplicación móvil.....	65
Tabla 6. Prueba de Aceptación 1 – Conexión Bluetooth y GPS.....	74
Tabla 7. Prueba de Aceptación 2 – Geo-localización y transmisión de datos OBDII.....	75
Tabla 8. Prueba de Aceptación 3 – Ejecución de la aplicación en segundo plano.	76
Tabla 9. Prueba de Aceptación 4 – Consulta de recorridos.....	76
Tabla 10. Prueba de Aceptación 5 – Subida de datos.	77
Tabla 11. Prueba de Aceptación 6 – Descarga de datos.	77

RESUMEN

Este trabajo describe la creación de una aplicación móvil Android para la extracción de datos de un vehículo a través de una interfaz de diagnóstico OBD-II, además de la geo-localización del vehículo por medio de un dispositivo móvil. Adicionalmente, se describe la creación de una aplicación web para descarga de los datos extraídos.

Para ello, se usa un scanner OBD-II, que permite la extracción y transmisión de datos de los sensores de un vehículo en tiempo real vía Bluetooth. En la aplicación móvil, se emplea el SDK de Android para: gestionar conexiones Bluetooth, obtener la ubicación por medio de GPS y determinar la conectividad del dispositivo para transmitir o almacenar los datos.

Adicionalmente, se usa el API de Google Maps para visualizar las ubicaciones obtenidas por la aplicación móvil mediante consultas, en una aplicación web basada en PHP que permite descargar los datos obtenidos en los formatos JSON y CSV.

Palabras Claves: OBD-II, datos, Android, API, Google Maps, PHP, CSV, JSON.

ABSTRACT

This paper describes the creation of a mobile application for Android extracting data from a vehicle through a diagnostic interface OBD-II, in addition to the geo-location of the vehicle via a mobile device. Additionally, creating a web application for discharge of the extracted data it is described.

For this, an OBD-II scanner, which allows the extraction and transmission of data from sensors of a vehicle in real time via Bluetooth, is used. In the mobile application, the Android SDK is used to: manage Bluetooth connections, get the location through GPS and determine connectivity device to transmit or store data.

Additionally, the Google Maps API is used to display the locations obtained by the mobile application by queries in a PHP based web application that lets you download data in JSON and CSV formats.

Keywords: OBD-II, data, Android, API, Google Maps, PHP, CSV, JSON.

INTRODUCCIÓN

Los automóviles actuales poseen multitud de sensores, los mismos que sirven para determinar el estado de un vehículo, por medio de interfaces de diagnóstico que usan estándares para su comunicación, como es la entrada de diagnóstico OBD-II.

Este trabajo tuvo como objetivo obtener los datos de un vehículo por medio de la interfaz de diagnóstico OBD-II a través del desarrollo de una aplicación móvil para la plataforma de Android, la misma que a su vez, obtiene la ubicación del vehículo por medio de geo-localización, para en conjunto transmitir los datos a un servidor o almacenarlos en el dispositivo móvil, dependiendo de si existe o no conectividad internet.

Los datos son almacenados en un servidor, el mismo que cuenta con una aplicación web para realizar consultas y graficar cada una de las ubicaciones obtenidas por medio de la geo-localización, así mismo ofrece la posibilidad de descargar la información obtenida del vehículo en formatos JSON y CSV.

El capítulo 1 describe conceptos relacionados con el origen de los sistemas de diagnóstico en los vehículos y el estándar OBD-II.

El capítulo 2 especifica el diseño de la solución, describiendo la arquitectura de la aplicación móvil así como de la aplicación web, además de especificar cada uno de sus módulos, funcionamiento y desarrollo basado en un enfoque iterativo.

El capítulo 3 comprende el desarrollo de los módulos de la aplicación móvil y web, basando el desarrollo e integración por iteraciones.

El capítulo 4 consta de la verificación del funcionamiento con un vehículo de prueba en un área determinada, así como pruebas de rendimiento tanto para la aplicación móvil como web.

Estos datos extraídos tienen múltiples usos, aplicabilidad para minería de datos y como base para otras aplicaciones y trabajos futuros.

La limitante principal es obtener el hardware OBD-II adecuado (scanner OBD-II) debido a que es un dispositivo genérico, existen muchas versiones y fabricantes del mismo. Así mismo, la precisión de la geo-localización está limitada a la capacidad hardware del dispositivo móvil, proveedores de servicios de ubicación y operadores de red en la zona en donde se ejecute la aplicación móvil.

La metodología usada es una adaptación de la metodología ágil XP (Xtreme Programming) ya que se usa su enfoque incremental para el desarrollo y pruebas funcionales.

CAPÍTULO 1: MARCO TEÓRICO

1.1. Introducción

1.1.1. Historia de los sistemas de diagnóstico a bordo.

En 1866 es el año en que Eugen Langen y Nikolaus Otto desarrollan un motor a gas, y 10 años más tarde en 1886 Otto construye el motor que sería la piedra angular para la evolución de los motores de combustión interna hasta los de nuestro tiempo: el motor de combustión interna de cuatro cilindros (Abrigo Maldonado 2007). Desde entonces, han pasado alrededor de 150 años y la industria vehicular ha dado avances significativos, lo que también ha originado consecuencias en el tema de contaminación emitida por la gran parte de los automóviles, mismas producidas por el uso de combustibles fósiles y desecho de gases.

1.1.2. Contaminación Vehicular

La contaminación vehicular se produce por la emisión de gases generados por la combustión del combustible, principalmente monóxido de carbono producido por la gasolina o diésel.

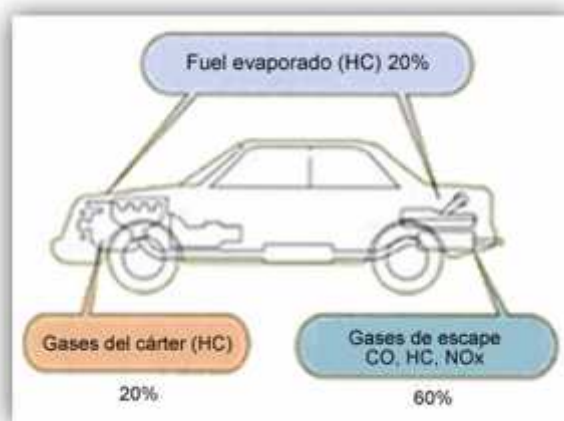


Figura 1. Elementos contaminantes de un vehículo.
Fuente: recuperado de <http://goo.gl/3KG7Pp>

Para combatir el problema de contaminación en la cuenca de Los Ángeles, el estado de California requirió sistemas de control de emisiones en los automóviles del modelo de 1966. El Congreso Norteamericano aprobó la ley de aire limpio en 1970 y se estableció la Agencia de Protección Ambiental (EPA)(Reitze 2001). Esta medida obligaba a los fabricantes a cumplir con estas normas, por ello crearon los sistemas de alimentación y encendido de combustible electrónicos, donde sensores miden el rendimiento del motor y ajustan los sistemas para reducir la contaminación(Cáceres and García 2012). Estos sensores también comenzaron a ser usados para diagnosticar el estado del vehículo.

1.2. Sistemas de diagnóstico a bordo: OBD

A finales de los años 1980, la California Air Resources Board (CARB) regularizó el uso de los Sistemas de Diagnóstico a Bordo (On-Board Diagnostics OBD)(EPA 2015b) para los vehículos comercializados en California a partir del año 1988.

Estos sistemas requerían que se identifiquen problemas relacionados con los sistemas de combustible, escapes, gases y sistemas relacionados con la unidad de control electrónico (Electronic Control Unit - ECU).

El estándar también establecía como requerimiento una alerta que se colocara en el tablero, un aviso de mal funcionamiento denominado MIL (Malfunction Indicator Lamp) el cual sugería al conductor que ocurría un problema y necesitaba ser revisado. Adicional a ello, internamente se almacenaba un código asociado al fallo DTC (Diagnostic Trouble Code) (Zaldivar et al. 2011) para una rápida identificación del problema.

Los fabricantes de vehículos implementaron dichas regulaciones, sin embargo hubo conflictos ya que cada fabricante tenía sus propios códigos de fallo y herramientas necesarias de identificación y lectura. La falta de homogenizar dichas implementaciones llevó a establecer una nueva regulación de estándares por parte de la CARB y la EPA.

1.3. Sistemas de diagnóstico a bordo versión II: OBDII

La CARB y la EPA crearon una nueva regulación que extendía la regulación original de los sistemas de diagnóstico a bordo, homogenizando procedimientos y estableciendo códigos comunes para facilitar la identificación en talleres de los errores detectados en los diferentes sistemas de emisión. Este estándar es el que se conoce actualmente como OBD-II.

OBD-II también marcó el estándar para el conector que se usaba para el diagnóstico, basado en señales eléctricas con protocolos estándar(Zaldivar et al. 2011). Este conector aparece en los vehículos desde el año de 1996.

Pin 2	J1850 Bus+
Pin 4	Chassis Ground
Pin 5	Signal Ground
Pin 6	CAN High (J-2284)
Pin 7	ISO 9141-2 K Line
Pin 10	J1850 Bus
Pin 14	CAN Low (J-2284)
Pin 15	ISO 9141-2 L Line
Pin 16	Battery Power




Figura 2. Entrada y pines OBDII

Fuente: recuperado de <https://goo.gl/xxhjAD>

El estándar de OBD-II implementa varios modos de trabajo; esto significa que, dependiendo la información a la que se desee acceder, se necesita un modo diferente. Una vez dentro de ese

modo de trabajo, se ofrece un extenso número de parámetros para acceder a dicha información(Meseguer 2012):

Tabla 1. Modos de trabajo del estándar OBDII.

Modo	Característica
01	Identificación de Parámetro (PID). Es el acceso a datos en vivo de valores de salidas y entradas a la ECU (Engine Control Unit).
02	Acceso a Cuadro de Datos Congelados. La ECU toma una muestra de todos los valores relacionados con las emisiones en el momento exacto de ocurrir un fallo.
03	Permite extraer de la memoria de la ECU todos los códigos de fallo (DTCs) almacenados.
04	Se pueden borrar todos los códigos almacenados en la PCM (Power Train Control Module) incluyendo los DTCs y el cuadro de datos grabado.
05	Devuelve los resultados de las pruebas realizadas a los sensores de oxígeno para determinar el funcionamiento de los mismos y la eficiencia del convertidor catalítico.
06	Permite obtener los resultados de todas las pruebas de abordó.
07	Permite leer de la memoria de la ECU todos los DTCs pendientes.

Nota. Fuente: Meseguer, Javier (2012). Publicado en: Caracterización de los estilos de conducción mediante Smartphones, dispositivos obd-ii y redes neuronales (p. 21). Valencia: Universidad Politécnica de Valencia.

Este estándar plantea beneficios para diversos tipos de usuarios:

- **Técnicos de reparación:** quienes pueden acceder de forma rápida a identificar los errores por medio de la lectura de códigos de fallos.
- **Gobiernos locales:** permite inspeccionar el cumplimiento de las leyes en favor del medio ambiente.
- **Propietarios de vehículos:** permite tener una alerta de los fallos en los sistemas del vehículo.
- **Empresas productoras de vehículos:** permite tener un control de sus motores y mejorar tecnologías para contrarrestar emisiones.

1.4. Aplicaciones móviles

Un sistema operativo móvil o SO móvil es un sistema operativo que controla un dispositivo móvil al igual que las computadoras utilizan Windows o Linux entre otros. Sin embargo, los sistemas operativos móviles son mucho más simples y están más orientados a la conectividad inalámbrica, los formatos multimedia para móviles y las diferentes maneras de introducir información en ellos(Aguirre Chacón and Sinche Ricra 2013). En la actualidad existen algunos sistemas operativos sobre las cuales se puede desarrollar aplicaciones.

1.4.1. Sistemas operativos móviles

Los sistemas operativos móviles más usados en la actualidad son:

- **ANDROID:** actualmente Android pertenece a Google, pero es un sistema abierto cualquier fabricante puede desarrollar en él sus productos.(Aguirre

Chacón and Sinche Ricra 2013) Actualmente se encuentra en la versión 6 denominada Marshmallow y es el sistema operativo más usado, debido a la variedad de dispositivos existentes que lo implementan.

- **IOS:** Es un sistema operativo móvil propietario, de la empresa Apple Inc. El cual sólo funciona en dispositivos fabricados por la propia empresa (iPhone). Actualmente se encuentra en la versión 9 y es el segundo sistema operativo más usado después de Android.
- **WINDOWS 10:** Lanzado en 2015 es el sistema operativo de los dispositivos de Microsoft, luego de abandonar su anterior sistema operativo Windows Mobile. No cuenta con una gran cuota de usuarios debido a la escasez de aplicaciones, sin embargo es un sistema integral ya que la misma versión se usa tanto en portátiles y PC como en Smartphone, Tablet.

1.5. Android OS

Android es un sistema operativo desarrollado por la Open Headset Alliance, compuesta por empresas del área tecnológica como compañías móviles incluyendo a China Mobile, HTC, Google, Qualcomm entre otras(Android Developers 2011). La filosofía que siguen es brindar un sistema operativo libre basado en la mejor experiencia de usuario.

1.5.1. Arquitectura de aplicaciones Android

La arquitectura de Android está dividida en 4 grandes capas: Aplicación, Framework de aplicaciones, librerías de SO y Kernel Linux(Brahler 2010).

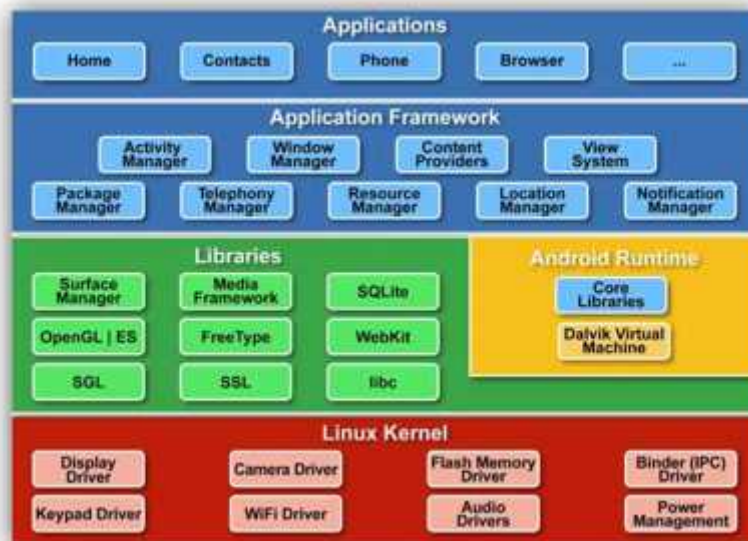


Figura 3. Arquitectura de Android OS.
Fuente: recuperado de <http://goo.gl/eHLX8z>

- **Capa de aplicación:** Son programas escritos en lenguaje Java que son ejecutadas en una máquina virtual.

- **Framework de aplicaciones:** Posee las APIs de Google, las cuales también pueden ser accedidas por los desarrolladores bajo las siguientes líneas de diseño:
 - Vistas escalables, incluidos elementos internos.
 - Proveedor de contenidos, que accede o comparte datos entre aplicaciones.
 - Gestor de recursos, como vistas o configuraciones.
 - Gestor de notificaciones, personalizadas por la aplicación.
 - Gestor de actividades, que determina el ciclo de vida de la aplicación.
 - Gestor de paquetes, ventanas.
- **Librerías de sistema operativo:** componentes soportados gracias al uso de C/C++ como son:
 - Media Framework, basado en la grabación-reproducción de múltiples formatos de audio, video e imagen.
 - SGL, motor de gráficos 2D.
 - SSL, protocolos TCP/IP con soporte de encriptación de comunicaciones de datos.
 - OpenGL, soporte de gráficos 3D.
 - SQLite, base de datos relacional.
 - Webkit, motor de navegación web.
- **Kernel Linux:** Los servicios de Android son basados en el kernel Linux como seguridad, procesamiento de datos, gestión de memoria y protocolos. También es la interacción entre hardware y software brindado controladores para pantalla, teclado, almacenamiento, cámaras, sonido, Bluetooth, WiFi y energía.

1.5.2. Aplicaciones Android

Google tiene una plataforma web con herramientas para desarrolladores. Actualmente, para comenzar a crear aplicaciones, primero se debe contar con la herramienta de desarrollo provista por la empresa, llamada Android Studio, un entorno de desarrollo basado en IntelliJ el cual ya posee el SDK (Software Development Kit) junto con la última librería de Android (otras para descarga) y todo lo necesario para empezar a codificar. También posee integración con otros entornos de desarrollo como Eclipse, a través de plugins.

1.6. Aplicaciones Web

Desde su concepción, la web tenía como objetivo primario el brindar información, sin embargo con la creciente evolución tecnológica esta ha ido evolucionando hasta convertirse en una fuente de interacción de aplicaciones. Estas aplicaciones web tienen características avanzadas, más experiencia de usuario, acceso multiplataforma, interacción con otros servicios y sistemas que proporcionan una mayor interacción.

“Una aplicación web es un sistema de software basado en tecnologías y estándares del Consorcio World Wide Web (W3C) que proporciona Web recursos específicos tales como contenidos y servicios a través de una interfaz de usuario, el navegador Web”(Kappel et al. 2006).

Así mismo, estas aplicaciones web tienen su propia complejidad basada en el contexto, es decir, puede ser muy básica como una web de información y contactos de una empresa hasta tan compleja como un sistema bancario.

Muchas de las arquitecturas web están basadas en un esquema de petición-respuesta, la comunicación síncrona establecida por un navegador hacia un servidor web por ejemplo, en donde pueden existir 3 componentes (Kappel et al. 2006):

- Cliente: Generalmente un navegador (o agente de usuario) es controlado por un usuario para utilizar la aplicación web.
- Servidor Web: Procesa las peticiones provenientes de los clientes, normalmente gestiona protocolos de comunicación e información.
- Servidor de base de datos: Gestiona la información de forma estructurada.

1.6.1. Desarrollo de aplicaciones web

Para el desarrollo de aplicaciones web se deben considerar aspectos relacionados al contexto, establecer objetivos y delimitar un alcance, esto permitirá determinar el grado de complejidad que requiere la solución. Desafortunadamente, no hay una vía común para estimar el tamaño de la aplicación, debido a la constante actualización de herramientas, tecnologías (Ruhe, Jeffery, and Wieczorek 2003) y en sí, de los requerimientos de la aplicación.

Existen varios tipos de arquitecturas que permiten el desarrollo de aplicaciones web, la más básica y conocida es la arquitectura de cliente-servidor, en soluciones más complejas se pueden aplicar arquitecturas basadas en n capas.

1.6.1.1. Arquitectura cliente-servidor

Esta arquitectura es la más básica para la ejecución de aplicaciones, implica la existencia de una relación entre procesos que solicitan servicios (*cliente*) y procesos que responden a estos servicios (*servidor*) (Luján Mora 2002). La principal característica es que es una arquitectura basada en niveles, por lo que se pueden separar funciones acorde a presentación, lógica y datos.

En el *nivel de presentación* se interactúa con el usuario, por medio de interfaces que permitirán la obtención de información y mostrar resultados, según sea el caso. El *nivel de lógica* tendrá almacenado todas las acciones orientadas con el procesamiento de la información, es decir la lógica de negocio. El *nivel de datos* se encarga del almacenamiento e integridad de los datos.

1.6.2. Tecnologías de desarrollo

Existen un sinnúmero de tecnologías que se pueden usar en combinación para crear aplicaciones web, entre las cuales se incluyen: lenguajes de programación, librerías, IDE's, frameworks, servidores y motores de bases de datos.

Las tecnologías más comunes de desarrollo son:

1.6.2.1. Lenguaje HTML

El lenguaje HTML (Hyper Text Markup Language) sirve para estructurar información en documentos (títulos, párrafos, listas, etc.) (Equipo Vértice 2009) dando un formato mediante etiquetas, las cuales poseen atributos y propiedades que ayudan a crear una

página web. Es el lenguaje soportado actualmente por todos los navegadores web multiplataforma.

1.6.2.2. Estilos CSS

Los estilos CSS determinan el aspecto visual de la información en un documento HTML (Van Lancker 2009). Los estilos pueden dar formatos a los caracteres con tamaños de letra, fuentes, colores, animaciones, tipos de cursores, etc. Al igual que HTML, tiene compatibilidad con los navegadores actuales, aunque haciendo uso de ciertas reglas dependiendo del navegador.

1.6.2.3. JavaScript

Este es un lenguaje basado en guiones que son interpretados directamente en el código HTML (Cobo et al. 2005). Debido a que es un lenguaje multiplataforma es usado en conjunto con otras tecnologías, pues siendo un lenguaje interpretado, no requiere ser compilado. Otra característica principal es que es orientado a objetos.

1.6.2.4. PHP

PHP es un lenguaje del lado del servidor, necesita un intérprete para su ejecución. PHP aporta dinamismo a las páginas HTML, la combinación de este lenguaje con el soporte que PHP tiene para tecnologías como JavaScript y comunicaciones con servidores de bases de datos lo hace uno de los lenguajes con más amplio uso y soporte.

1.6.2.5. Servidor de Datos MySQL

MySQL es un servidor de datos de código abierto, posee soporte multiplataforma el cual le permite interactuar con diferentes lenguajes y tecnologías. Su uso es amplio en entornos educativos y también en pequeñas empresas.

CAPÍTULO 2: DISEÑO DE LA SOLUCIÓN

2.1. Introducción

Mucha de la información suministrada por los sensores en un vehículo no es realmente aprovechada, existen datos que son generados a cada momento mientras se usa el vehículo, sin que sean usados o aplicados para estimar el estado del vehículo en determinado momento.

En el panel (tablero) del conductor se muestran datos sobre revoluciones por minuto (RMP), velocidad, temperatura, kilometraje y algunas advertencias como chequeo del motor en caso de algún fallo. Estos datos para el propietario son suficientes para conocer el estado del vehículo y establecer cuándo debe realizar el mantenimiento. Si un problema surge, debe acudir al centro de servicio autorizado, muchas de las veces, desconociendo el posible origen del problema.

Sin embargo, existen otros datos además de los mencionados. El uso de estos datos estadísticos podrían por ejemplo, estimar la cantidad de gasolina consumida en cierto tiempo para poder establecer un plan de ahorro de combustible o determinar patrones de consumo excesivo de combustible en función del estilo de conducción aplicado, incluso valiéndose de datos de ubicación para una mejor estimación.

Entonces surge la necesidad de recolectar estos datos, que después de ser procesados, podrían ayudar a estimar de mejor forma el estado del vehículo y el uso que se ejerce sobre el mismo, pudiendo incluso estimar futuras anomalías.

2.2. Solución Propuesta

2.2.1. Objetivo General

Desarrollar una aplicación móvil en el sistema operativo Android para lectura, transmisión y almacenamiento de los datos generados por un vehículo a través de la entrada estándar OBD-II, y obtención de su ubicación a través de geo-localización.

2.2.2. Objetivos Específicos

- Identificar el hardware adecuado para la extracción de datos desde el vehículo.
- Identificar incompatibilidades entre hardware/software de dispositivos y vehículos.
- Determinar los datos más relevantes que se deben extraer.
- Obtener la ubicación del vehículo para graficar su posición en tiempo real.
- Almacenamiento y descarga de los datos extraídos.

2.3. Descripción de la Solución

Para realizar la extracción de datos del vehículo, se hará uso de la entrada OBD-II descrita anteriormente, que a través de un hardware compatible se comunicará con el aplicativo móvil Android para la transmisión de datos, adicionando también la ubicación del vehículo por geo-localización con el GPS del dispositivo móvil. Los datos serán almacenados en un servidor, para su posterior gestión.



Figura 4. Flujo de la solución propuesta.
Fuente: el autor.

Dada la diversidad de modelos y marcas de vehículos, se pretende usar como objeto de prueba un auto familiar de uso común.

Se considera el siguiente esquema de solución, que contiene varios componentes:

- Hardware OBDII, que comprende el uso de un scanner OBDII que sirve para la transmisión de datos desde el vehículo.
- Aplicación móvil Android, para la recolección, envío/almacenamiento de la información generada desde el vehículo a través del scanner OBDII y su geo-localización, la que obtendrá la ubicación del vehículo por medio del GPS del dispositivo móvil Android.
- Servidor de datos, que almacenará los datos transmitidos por la aplicación.
- Aplicación web de visualización/exportación de datos.



Figura 5. Esquema de la solución.
Fuente: El autor.

2.3.1. Hardware OBD-II

OBD-II Monitorea el comportamiento de los sistemas de emisión y de los componentes, así como los fallos eléctricos y almacena información para un uso posterior (Meseguer 2012).

El principal obstáculo a la hora de elegir un scanner OBD-II es la variedad del tipo de interfaces y “marcas blancas” debido a que este dispositivo es genérico. Principalmente existen 4 tipos de scanner:

- WiFi: Esta interfaz origina un punto de red wifi al cual conectarse para la extracción de datos.
- Bluetooth: Hay que vincular el dispositivo vía Bluetooth, al vincular pedirá una clave que por lo general es 1234 o también 0000.
- USB: Desde el scanner se conecta un cable USB que permite conectarse mediante un puerto serial.
- Mixto: Pueden incluir interfaz USB/WiFi en un mismo dispositivo, algunos incluso incorporan las 3 conexiones.

2.3.1.1. Selección del scanner OBDII.

Para realizar la selección del scanner apropiado, se realizaron dos pruebas de hardware las mismas que sirvieron para identificar el tipo de scanner apropiado para trabajar en el entorno de Android.

2.3.1.1.1. Prueba 1.

En cuanto a hardware, se seleccionó el scanner OBD-II cuyas características son:

- Marca: Marca blanca (genérico).
- Tipo: Mixto.

- Conexiones: WiFi, USB.



Figura 6. Scanner OBDII Wifi-USB.
Fuente: recuperado de <http://goo.gl/H3NTzu>

Se proporcionó por parte del Departamento de Inteligencia Artificial de la Universidad Técnica Particular de Loja el siguiente dispositivo:

- Marca: Samsung.
- Modelo: Galaxy Tab 3.
- Tipo: Tablet.
- Conexiones: Bluetooth 4.0, WiFi, GPS, 3G-HSPDA.
- Sistema Operativo: Android 4.1 Jelly Bean.



Figura 7. Samsung Galaxy Tab 3.
Fuente: recuperado de <http://goo.gl/s9IICT>

Para realizar las pruebas de funcionamiento del scanner, se usó la aplicación "Torque" disponible en la tienda oficial de aplicaciones de Android Google Play.



Figura 8. Aplicación Torque.
Fuente: recuperado de <http://goo.gl/0QJDe6>

Se seleccionó el siguiente vehículo:

- Marca: Chevrolet.
- Modelo: Corsa Evolution.
- Año: 2009.

2.3.1.1.2. Resultados



Figura 9. Pruebas del scanner WiFi-USB en modo WiFi.
Fuente: el autor.

Realizadas las pruebas con los dispositivos, se encontró que el dispositivo Android no mostraba la conexión para vincular vía WiFi para acceder al scanner OBD-II.

Para descartar el mal funcionamiento del scanner OBD-II, se hizo pruebas con su conexión USB con un PC de las siguientes características:

- Marca: Acer.
- Modelo: Aspire 4738.
- Procesador: Intel i3 Primera Generación.
- 4 GB de RAM, 500 GB Disco.

- Sistema: Windows 7.
- USB 2.0



Figura 10. Pruebas del scanner WiFi-USB en modo USB.
Fuente: el autor.

La conexión fue exitosa, para verificar que la interfaz WiFi no tuviera problemas, se usó un dispositivo iOS de las siguientes características:



Figura 11. Pruebas con el scanner WiFi-USB con iPad.
Fuente: el autor.

- Marca: Apple.
- Modelo: iPad 4
- Conexiones: Bluetooth 4.0, WiFi, GPS, 3G-HSPDA+

La interfaz WiFi trabajó sin problemas, por lo que se determinó que dicho scanner no era el apropiado para trabajar con el sistema operativo Android, ya que no era compatible vía WiFi.

2.3.1.1.3. Prueba 2.

Para esta prueba, se adquirió un scanner Bluetooth, el mismo que tiene las siguientes características:



Figura 12. Scanner OBDII Bluetooth.
Fuente: recuperado de <http://goo.gl/1Rv14U>

- Marca: HH.
- Modelo: OBD Advanced.
- Tipo: Interfaz única.
- Conexiones: Bluetooth.

2.3.1.1.4. Resultados

Ejecutando las pruebas de conexión con el scanner, el dispositivo y el vehículo antes mencionados, se conectó transmitiendo los datos de forma exitosa. Por lo tanto, para esta investigación se seleccionó un scanner con una interfaz Bluetooth. El proveedor de este scanner en particular, señala la compatibilidad con vehículos desde el año 2003 en adelante.



Figura 13. Pruebas del scanner Bluetooth en la Tablet Tab 3.
Fuente: el autor.

2.3.2. Aplicación Móvil Android

Para el desarrollo de la aplicación, teniendo en cuenta que se ha seleccionado el scanner OBD-II versión Bluetooth como medio de lectura de datos, es necesaria la interacción con el adaptador Bluetooth en el entorno Android, para lo cual el sistema operativo ofrece los métodos necesarios en el API para desarrolladores(Google 2016a) en su página oficial.

Alineado a los objetivos propuestos, la aplicación debe comunicarse vía Bluetooth con el scanner OBDII para la lectura de datos, obteniendo en simultáneo la ubicación mediante geo-localización y por último, dependiendo del estado de conectividad de red del dispositivo, transmitir o almacenar en un archivo los datos obtenidos.

2.3.2.1. Arquitectura de la aplicación

Se puede establecer la arquitectura de la aplicación en base al siguiente esquema:

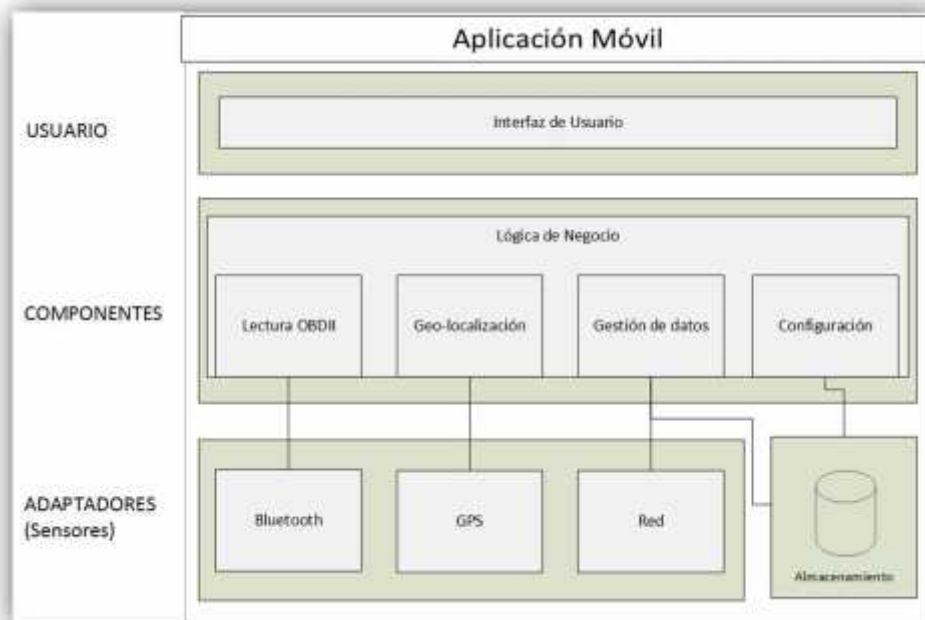


Figura 14. Arquitectura de la aplicación móvil.
Fuente: el autor.

En base a la arquitectura, podemos definir los módulos de la aplicación:

2.3.2.2. Módulo de Lectura OBD-II

Este módulo comprende la comunicación vía Bluetooth con el scanner OBD-II, desde el establecimiento de la conexión, envío de comandos OBD y recepción de la respuesta.

Para la comunicación, se considera usar el API de desarrollo para gestionar el adaptador Bluetooth del dispositivo, con lo cual se tiene acceso a: activar el Bluetooth, lista de dispositivos vinculados, dispositivos disponibles y la conexión hacia algún dispositivo en particular, por medio del nombre y dirección MAC.

Para la lectura OBD-II las APIs proporcionan un método de interacción, el cual mediante una conexión “Bluetooth Serial Board” (Google 2016a) establece un socket de comunicación de 2 vías en el cual se envían los comandos OBDII en tipo carácter y se recibe una respuesta de tipo hexadecimal, que con el debido procesamiento en la aplicación web dará el dato en un formato legible. No se realiza este procesamiento en el dispositivo móvil para disminuir el consumo de recursos de la aplicación, permitiendo que la aplicación funcione de forma transparente para el usuario, ya que tendrá que ejecutarse de forma recurrente en pequeños lapsos de tiempo, este parámetro será configurado por el usuario.

El siguiente diagrama de flujo representa la gestión del módulo de lectura OBD-II.

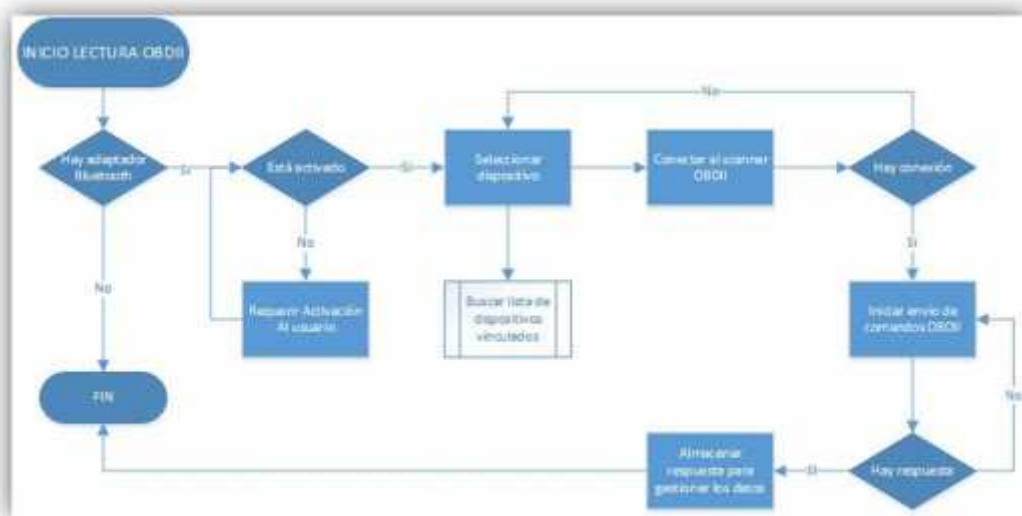


Figura 15. Diagrama de flujo del módulo de lectura OBD-II.
Fuente: el autor.

2.3.2.3. Módulo de Geo-localización

Este módulo obtiene la ubicación a través de las coordenadas de latitud y longitud por medio del adaptador GPS integrado en el dispositivo móvil. El API de desarrollo posee métodos para obtención de estos datos(Google 2016b), se hace uso de un proveedor de servicios de ubicación, asistido también por el geo-posicionamiento con redes móviles (WiFi/datos). La exactitud de los datos dependerá exclusivamente de los proveedores de ubicación/red y de la capacidad hardware del dispositivo móvil.

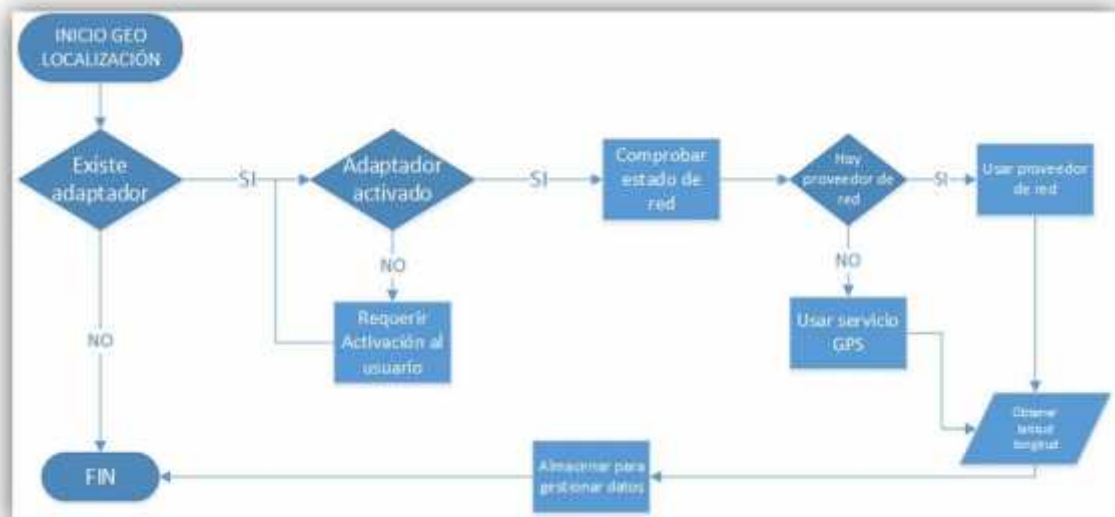


Figura 16. Diagrama de flujo del módulo de geo-localización.
Fuente: el autor.

2.3.2.4. Módulo de Gestión de datos

El módulo de gestión de datos se encarga del manejo de los datos obtenidos tanto por el scanner OBD-II como por el dispositivo móvil (geo-localización).

Para este módulo, primero se debe tener el conjunto de datos obtenidos tanto desde el vehículo como del dispositivo móvil Android, estos datos servirán de entrada para ser gestionados dependiendo del estado de conexión de red del dispositivo móvil: si existe conexión internet, los datos serán enviados directamente hacia el servidor para ser almacenados, si no existe conexión, los datos se almacenarán en un archivo ubicado dentro del almacenamiento interno del dispositivo, en un directorio llamado "OBD2". El formato de almacenamiento será CSV.

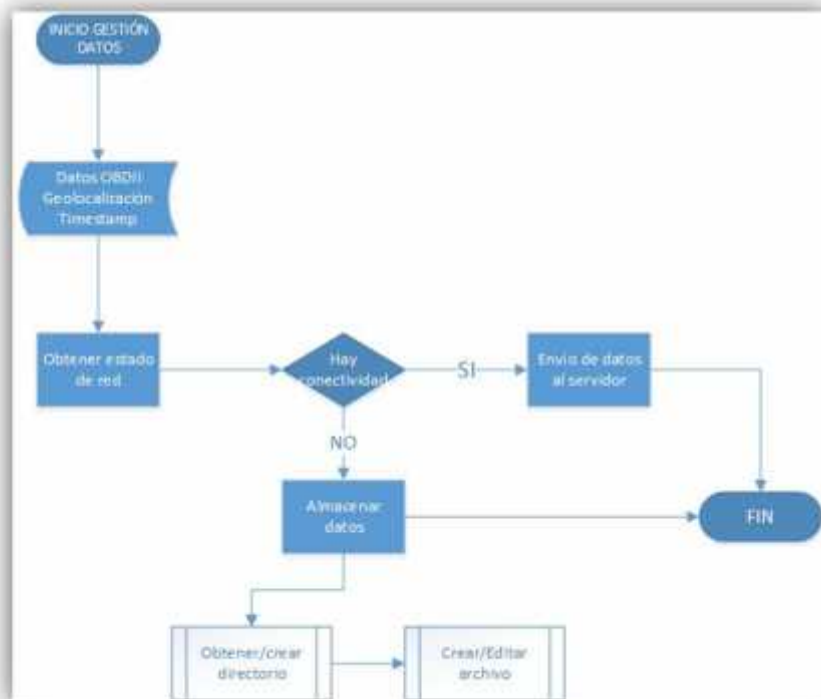


Figura 17. Diagrama de flujo del módulo de gestión de datos.
Fuente: el autor.

El archivo CSV almacenará los datos separados por comas, con la siguiente estructura:

- FECHA: timestamp de la fecha de recolección de los datos.
- TIEMPO: timestamp de la hora, minuto y segundo de la recolección de datos.
- DESCRIPCION: la descripción del dato extraído (ejemplo: “velocidad”).
- COMANDO: el comando OBDII enviado (ejemplo: “012F”).
- LATITUD: coordenada de latitud expresada en decimales.
- LONGITUD: coordenada de longitud expresada en decimales.
- DATO: el dato extraído del vehículo, expresado en hexadecimal (ejemplo: “41010E”).
- IDENTIFICADOR: el ID del usuario/vehículo configurado en la aplicación móvil.

2.3.2.5. Módulo de Configuración

El módulo de configuración se encarga de obtener los parámetros configurables por el usuario.

El usuario de la aplicación será capaz de ajustar por medio de estos parámetros:

- El periodo de tiempo entre cada lectura de los datos desde el vehículo.
- Un ID que permita identificar al usuario/vehículo del cual provienen los datos.
- La dirección del servidor hacia dónde enviar los datos.

Estos parámetros los podrá ajustar manualmente por medio de un archivo de configuración ubicado en el almacenamiento interno del dispositivo móvil.

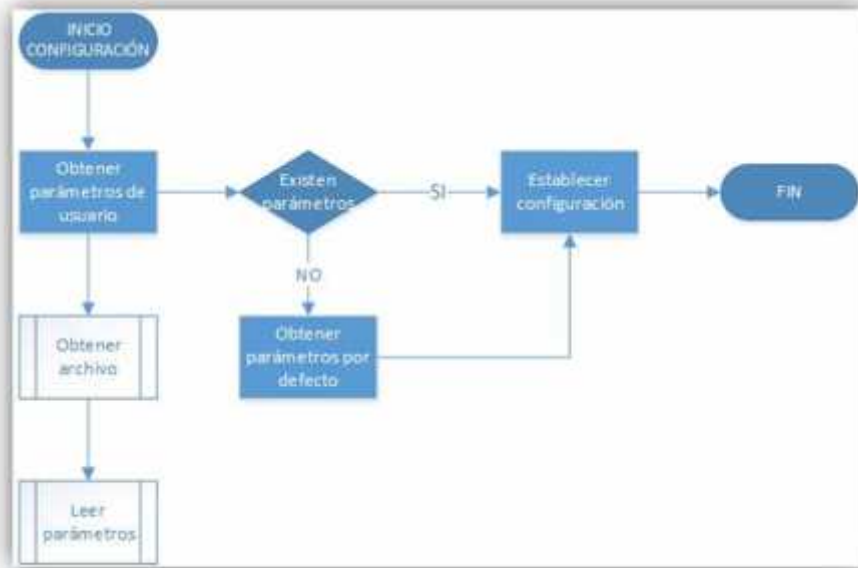


Figura 18. Diagrama de flujo del módulo de configuración.
Fuente: el autor.

Adicionalmente, si el archivo de configuración no se encuentra o no tiene los parámetros establecidos, se toma una configuración por defecto que se encuentra dentro de la programación de la aplicación, la cual establece:

- Periodo de lectura de datos: 180 segundos
- Identificador: test
- Dirección del servidor: <http://tesisobd.herokuapp.com/procesar.php>

2.3.2.6. Servidor de Datos

Dentro de la gestión de datos, se necesita almacenar la información transmitida por la aplicación móvil.

Se hará uso de tecnología de SAS (Software as Service) con el uso de una base de datos cloud como es ClearDB, un servicio de base de datos en la nube para aplicaciones con base en MySQL(Heroku 2016a).

Dado que la estructura de almacenamiento no es compleja, el modelo de base de datos solo requiere una tabla en la cual se almacenará los datos transmitidos desde la aplicación móvil.


Name	Type	Length	Decimals	Not null	
ID	int	11	0	<input checked="" type="checkbox"/>	 1
FECHA	varchar	250	0	<input type="checkbox"/>	
TIEMPO	varchar	250	0	<input type="checkbox"/>	
DESCRIPCION	varchar	500	0	<input type="checkbox"/>	
COMANDO	varchar	250	0	<input type="checkbox"/>	
LATITUD	varchar	250	0	<input type="checkbox"/>	
LONGITUD	varchar	250	0	<input type="checkbox"/>	
DATO	varchar	250	0	<input type="checkbox"/>	
IDENTIFICADOR	varchar	255	0	<input type="checkbox"/>	

Figura 19. Estructura de la base de datos.
Fuente: el autor.

La tabla contiene los siguientes campos:

- ID: entero generado automáticamente en cada inserción de datos, es la llave primaria.
- FECHA: timestamp de la fecha de recolección de los datos.
- TIEMPO: timestamp de la hora, minuto y segundo de la recolección de datos.
- DESCRIPCION: la descripción del dato extraído (ejemplo: “velocidad”).
- COMANDO: el comando OBDII enviado (ejemplo: “012F”).
- LATITUD: coordenada de latitud expresada en decimales.
- LONGITUD: coordenada de longitud expresada en decimales.
- DATO: el dato extraído del vehículo, expresado en hexadecimal (ejemplo: “41010E”).
- IDENTIFICADOR: el ID del usuario/vehículo configurado en la aplicación móvil.

2.3.3. Aplicación Web

La aplicación web servirá para cargar, visualizar y exportar la información obtenida desde la aplicación móvil.

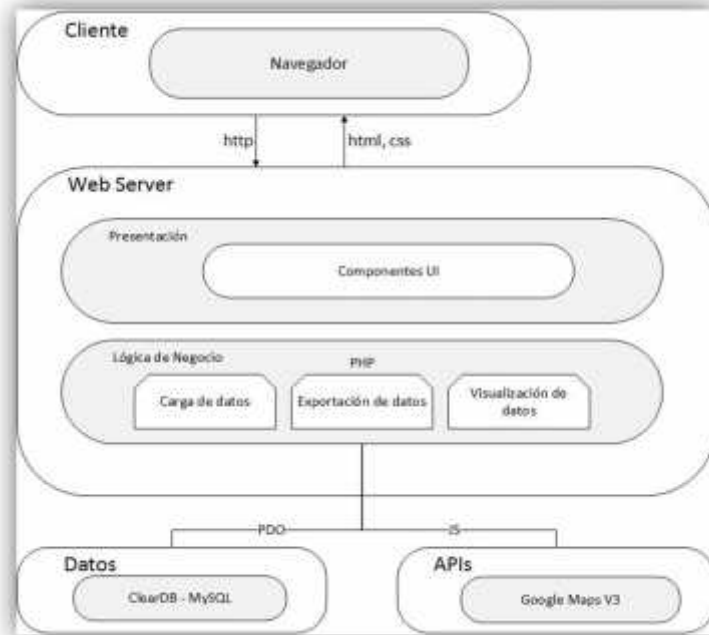


Figura 20. Arquitectura de la aplicación web.
Fuente: el autor.

La aplicación web consta de tres módulos para gestionar el almacenamiento, visualización y exportación de datos:

2.3.3.1. Módulo de Carga de datos

Este módulo permite la carga de un archivo CSV que es generado por la aplicación móvil cuando el dispositivo no posee conexión. Este archivo contiene los datos con la estructura descrita anteriormente¹ y se encuentra almacenado en el almacenamiento interno del dispositivo móvil.

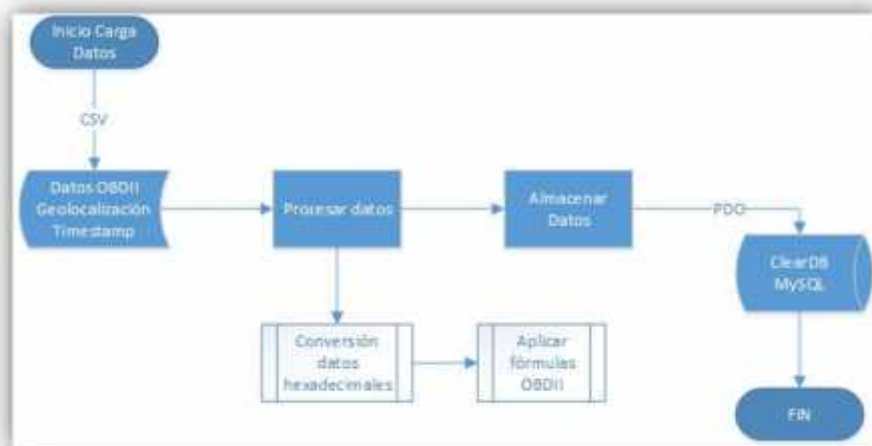


Figura 21. Diagrama de flujo del módulo web de carga de datos.
Fuente: el autor.

¹ Ver Módulo de Gestión de Datos.

Primero, los datos contenidos en el archivo CSV deben procesarse ya que los datos de la lectura OBDII están almacenados en formato hexadecimal².

Después de la conversión, dependiendo del tipo de comando OBD se aplica una fórmula para calcular el dato real en formato legible(EPA 2015a).

Seguidamente, se comunica con la base de datos en la nube (ClearDB)(Heroku 2016a) para almacenar los datos.

2.3.3.2. Módulo de Visualización de datos

Este módulo permite visualizar los datos de latitud, longitud en un rango de fecha y tiempo determinados, usando el API de Google Maps V3(Google Developers 2016a).

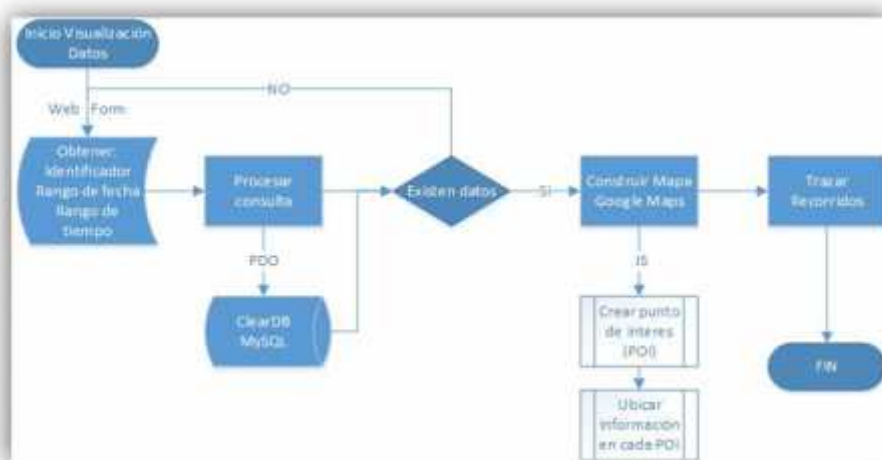


Figura 22. Diagrama de flujo del módulo web de visualización de datos.
Fuente: el autor.

Se obtienen los datos de entrada que servirán para elaborar la consulta: identificador, el rango de fechas y horas para especificar de mejor manera la información a consultar. Se realiza la consulta hacia la base de datos en la nube la cual en caso de obtener resultados, devuelve los valores hacia el módulo de visualización, el cual mediante el API de Google Maps V3 presenta los datos de latitud y longitud en forma de puntos de interés (POI)(Google Developers 2016b) en el mapa, adicionalmente se muestra los datos de fecha y hora en cada POI.

Una vez representados los POI, se procede a trazar la ruta recorrida, es decir una línea que interconecta cada POI dentro del mapa, de forma preliminar en orden cronológico. Cabe señalar que este módulo trabaja en forma conjunta con el módulo de exportación de datos ya que hace uso de la misma consulta, descrita anteriormente.

² Ver Módulo de Lectura OBD-II.

2.3.3.3. Módulo de Exportación de datos

Este módulo permite obtener todos los datos consultados a través del módulo web de visualización en formatos consumibles por otras aplicaciones, como son JSON y CSV.

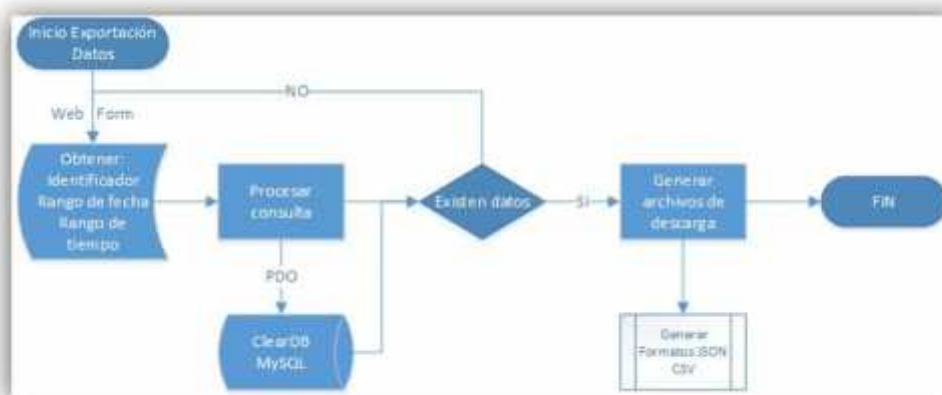


Figura 23. Diagrama de flujo del módulo web de exportación de datos.
Fuente: el autor.

Este módulo trabaja en paralelo con el módulo anteriormente descrito³, ya que recibe como datos de entrada el identificador, el rango de fechas y horas, los mismos que se usan para realizar la construcción del mapa de Google Maps V3.

2.4. Alcance de la solución

El desarrollo de esta aplicación, tanto en la aplicación móvil como en la aplicación web de visualización de datos, está limitado acorde a los siguientes puntos:

2.4.1. Aplicación Móvil Android

- Desarrollada para dispositivos con el sistema operativo Android desde la versión 4.0 (Ice Cream Sándwich) y probada hasta la versión 6.0.1 (Marshmallow).
- La configuración de la aplicación será ingresada manualmente por el usuario quien tendrá que crear un archivo de configuración en el almacenamiento interno del dispositivo móvil.
- No se contempla realizar pruebas de compatibilidad de la aplicación móvil en la versión 7 (Android N) debido a que dicha versión (a la fecha de realización de esta aplicación) aún no se encuentra oficialmente disponible por los fabricantes en los dispositivos que se dispone para la realización de pruebas de funcionamiento.
- Dispositivos que cuenten con hardware Bluetooth versión 2.1 o superior.
- Dispositivos con GPS integrado. La exactitud de los datos de geo-localización dependen exclusivamente de la capacidad hardware del dispositivo y los proveedores de ubicación disponibles en el área de pruebas.
- No se contempla la distribución de la aplicación en la tienda de aplicaciones oficial de Android (Google Play).

³ Ver Módulo Web de Visualización de Datos.

2.4.2. Aplicación Web

- La dirección para el acceso será un sub-dominio. Las limitaciones dependen exclusivamente del proveedor de hosting.
- Desarrollada bajo el lenguaje PHP y tecnologías compatibles como HTML5, JavaScript, Bootstrap. Las limitaciones dependen exclusivamente del proveedor de hosting.
- Servidor apache y extensiones habilitadas para soportar el intérprete de lenguaje PHP. Las limitaciones dependen exclusivamente del proveedor de hosting.
- Motor de base de datos MySQL. Las limitaciones dependen exclusivamente del proveedor del servicio.
- Restricciones propias del API de Google Maps para la visualización de las ubicaciones y recorridos.

2.4.3. Vehículos

- Se realizarán las pruebas en un vehículo liviano que disponga de la entrada estándar OBD-II.
- Existen limitaciones propias del fabricante del vehículo en cuanto a protocolos de comunicación de sus sensores internos y la Unidad de Control Eléctrico (ECU) que dependen de la marca, modelo, año y mercado de destino.
- En base al punto anterior, se limitarán las pruebas a vehículos cuyo modelo sea del año 2015 en adelante.

2.4.4. Hardware OBDII

- Se seleccionará el hardware compatible para la transmisión de datos con la aplicación móvil Android (scanner Bluetooth).
- Dado que el hardware OBD-II es muy genérico, aplican restricciones de compatibilidad del scanner respecto a los vehículos debido a su construcción (circuitos internos)⁴.

2.4.5. Otras consideraciones

- El área geográfica para las pruebas de la aplicación está limitado a la ciudad de Loja, Ecuador, sujeto a la disponibilidad del uso del vehículo.
- La transmisión de datos y la geo-localización dependen de la cobertura del operador móvil, proveedores de servicios de ubicación y del hardware del dispositivo Android.

2.5. Metodología de Desarrollo

El desarrollo de este trabajo se llevará en base a una metodología ágil, por lo que se usarán algunos principios de la metodología XP (Extreme Programming).

2.5.1. Extreme Programming XP

XP es una nueva metodología de desarrollo: ligera, pero eficiente considerando riesgos, flexible, predecible, enfocada en el desarrollo y orientada a muchos cambios.

La programación extrema es inventada por K. Beck(Shi et al. 2011). Se inicia por recoger las historias de usuario para poder estimar los requerimientos, se codifica, se

⁴ Ver Anexo 3. Hardware y circuitos OBD-II.

generan los paquetes de software que son probados y posteriormente integrados al sistema final.

Su enfoque de planificación incremental y la retroalimentación continua de los ciclos cortos del desarrollo de software permiten un proceso de diseño evolutivo que dura tanto como el sistema en desarrollo (Juric 2000). Permite la comunicación constante, lo que ayuda a detectar errores de forma temprana.

2.5.1.1. Fases de XP

Al ser una metodología ágil, XP posee 4 fases establecidas:



Figura 24. Fases de la metodología XP.
Fuente: recuperado de <http://goo.gl/em7QwN>

- **Planificación:** Es una fase que consta de reuniones entre el cliente y los desarrolladores, en donde se definen los requerimientos y se documentan como “historias de usuario”. El resultado es la visión general del sistema y la prioridad de cada historia de usuario.
- **Diseño:** Se define una metáfora del sistema, creando la funcionalidad mínima y cómo el sistema debe comportarse, en pocas palabras, un diseño simple.
- **Desarrollo:** Se codifica el sistema de forma iterativa, se genera al final de cada una un entregable funcional el cual refleja el resultado de una o más historias de usuario. También la integración constante es parte fundamental de esta fase.
- **Pruebas:** Se revisa la integridad y funcionalidad del sistema, aplicando pruebas unitarias y de aceptación.

2.5.2. Plan de Entregas

Teniendo en cuenta las fases de XP, podemos definir un plan de entregas en base a iteraciones de la siguiente forma:

Tabla 2. Plan de Iteraciones

Módulo	Iteración	Actividad	Tipo
Geo-localización	1	Obtención de proveedores de servicios de ubicación.	App. Móvil
	2	Obtención de ubicación.	
Lectura OBDII	1	Administrar conexiones Bluetooth.	App. Móvil
	2	Envío de comandos OBDII.	
Gestión de Datos	1	Determinar el estado de conectividad del dispositivo móvil.	App. Móvil
	2	Creación del directorio de almacenamiento.	
		Creación del archivo CSV. Subida de datos al servidor.	
Configuración	1	Lectura de parámetros del usuario.	App. Móvil
Carga de Datos	1	Modelo de base de datos	App. Web
	2	Diseño de interfaz para subida del archivo CSV. Almacenamiento de datos desde la aplicación móvil.	
Visualización de Datos	1	Diseño de interfaz para consulta.	App. Web
	2	Diseño de interfaz de visualización.	
		Consumo del API Google Maps.	
		Mostrar marcadores con la ubicación. Trazar rutas entre marcadores.	
Exportación de Datos	1	Generar archivo CSV.	App. Web
		Generar archivo JSON.	

Nota Fuente: el autor.

2.6. Herramientas de desarrollo

Para el desarrollo se ha contemplado el uso del sistema operativo Windows 10 con los siguientes recursos:

Tabla 3. Herramientas de Desarrollo.

Herramienta	Característica
Eclipse Mars IDE	Eclipse es un entorno de desarrollo multiplataforma, proporciona muchas funcionalidades para codificación y pruebas.
Android SDK	Es el kit de desarrollo de Google para la plataforma de Android, se usará de base la versión 4 para compilar la aplicación.
Android Development Tool (ADT) Plugin para Eclipse	Complemento para el IDE Eclipse, que permite hacer uso de las librerías y herramientas del Android SDK.
Heroku Toolbelt	Gestor de comandos para interactuar con el servicio de Hosting de Heroku, para la aplicación web.
Navicat Premium	Herramienta para administración de bases de datos en la nube.

Twitter Bootstrap 3	Librerías para el diseño de interfaces web responsivas.
Sublime Text 3	Editor de código.
Sybase Power Designer 15	Herramienta para modelado de negocio y de base de datos.

Nota Fuente: el autor.

2.6.1. Hosting, Dominio: Heroku

Para el desarrollo se ha contemplado el uso de la plataforma de Heroku(Heroku 2016b), posee almacenamiento de aplicaciones en la nube, con soporte para múltiples tecnologías incluyendo PHP, Python, Java, Ruby entre otras.⁵

2.6.2. Servidor de Base de Datos: ClearDB

Al usar Heroku como hosting brinda la posibilidad de desplegar aplicaciones bajo el subdominio herokuapp.com así mismo permite el uso de addons o plugins para extender los servicios de las aplicaciones, también se hará uso del addon Heroku ClearDB(Heroku 2016a) como motor de base de datos.

⁵ Ver Anexo 2. Hosting de Aplicaciones en la plataforma de Heroku.

CAPÍTULO 3: DESARROLLO

3.1. Proyecto en Eclipse

Como se define en el plan de entregas, el desarrollo se llevará en 4 iteraciones, se explican en función de módulos tanto para la aplicación móvil como la aplicación web, cabe destacar que tal como se muestra en la tabla 2 descrita al término del capítulo anterior⁶, algunos módulos se trabajarán en paralelo.

Para el inicio del desarrollo de la aplicación móvil, se crea un nuevo proyecto en Eclipse(Eclipse Foundation 2016), para lo cual previamente se debe tener instalado el plugin Android Developer Tool (ADT)(Android Developers 2016a), el SDK de Android(Android Developers 2016b) y dentro del SDK, las APIs de desarrollo para la versión que se quiera desarrollar la aplicación, la versión 4 en este caso.

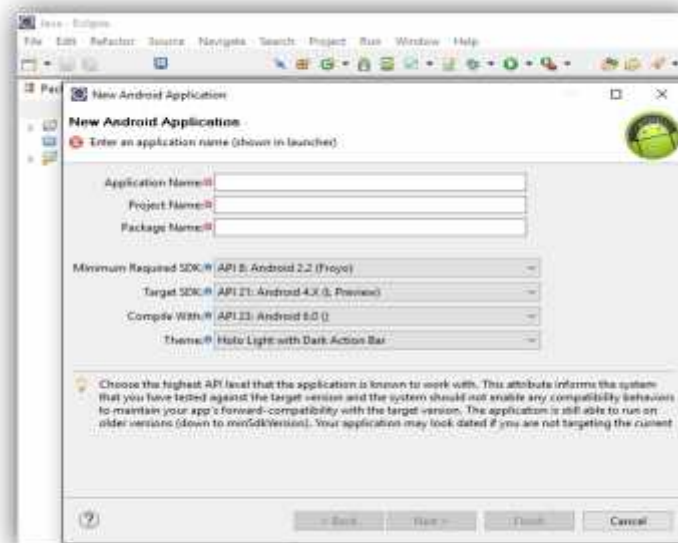


Figura 25. Creación de un proyecto de Android en Eclipse IDE.
Fuente: el autor.

Por defecto, el ADT y el SDK generan la estructura necesaria para el desarrollo de la aplicación, tal como se muestra a continuación en el proyecto “OBDTEST”:

⁶ Ver Tabla 2. Plan de Iteraciones.

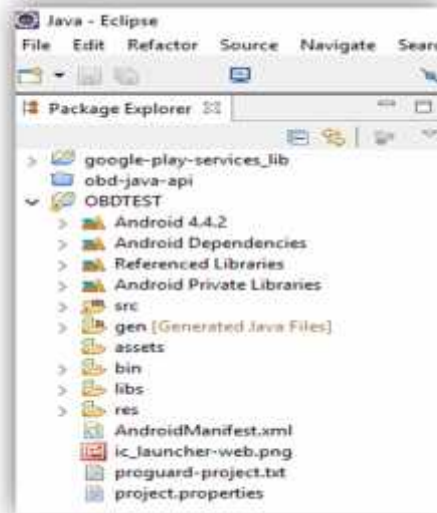


Figura 26. Estructura de archivos del proyecto Android.
Fuente: el autor.

3.2. Módulo de geo-localización

3.2.1. Iteración 1.

Para el módulo de geo-localización se debe crear la clase correspondiente, primero se tratará de conseguir la ubicación por medio de proveedores de red y en segunda instancia, se hará uso del GPS.

3.2.1.1. Clase geo.java

Esta clase permitirá obtener las coordenadas de ubicación. La clase se implementa bajo las invocaciones de herencia de *Service* y *LocationListener*, esto hará que se pueda usar el API de control de servicios de ubicación disponibles en el dispositivo móvil.

Según la documentación del API(Google 2016b) Se puede hacer uso de dos parámetros adicionales que son el tiempo de actualización y la distancia, los mismos que servirán para actualizar la ubicación durante algún periodo definido.

```
// The minimum distance to change Updates in meters
private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 10; // 10 meters

// The minimum time between updates in milliseconds
private static final long MIN_TIME_BW_UPDATES = 1000 * 60 * 1; // 1 minute
```

Figura 27. Módulo de Geo-localización - declaración de parámetros.
Fuente: recuperado de <http://goo.gl/9PIaU9>

Una vez definidos, se procede a crear el método para obtener la ubicación, es importante en este punto obtener los proveedores de ubicación, ya sea por red o GPS que nos permitirán obtener posteriormente las coordenadas de latitud y longitud.

Se definen variables booleanas para establecer el estado de red y del GPS, dependiendo de su estado, se invocará el método para obtener las coordenadas.

```
public Location getLocation() {
    try {
        locationManager = (LocationManager) mContext
            .getSystemService(LOCATION_SERVICE);

        // getting GPS status
        isGPSEnabled = locationManager
            .isProviderEnabled(LocationManager.GPS_PROVIDER);

        // getting network status
        isNetworkEnabled = locationManager
            .isProviderEnabled(LocationManager.NETWORK_PROVIDER);
    }
}
```

Figura 28. Módulo de Geo-localización - Estado de red y GPS.
Fuente: recuperado de <http://goo.gl/9PIaU9>

Obtenido el estado de red y GPS y dependiendo si está activo o no, se procede a enviar la petición al servicio de ubicación.

```
if (isNetworkEnabled) {
    locationManager.requestLocationUpdates(
        LocationManager.NETWORK_PROVIDER,
        MIN_TIME_BW_UPDATES,
        MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
}
```

Figura 29. Módulo de Geo-localización - Ubicación vía red.
Fuente: recuperado de <http://goo.gl/9PIaU9>

```
if (isGPSEnabled) {
    if (location == null) {
        locationManager.requestLocationUpdates(
            LocationManager.GPS_PROVIDER,
            MIN_TIME_BW_UPDATES,
            MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
    }
}
```

Figura 30. Módulo de Geo-localización - Ubicación vía GPS.
Fuente: recuperado de <http://goo.gl/9PIaU9>

3.2.2. Iteración 2.

Para esta segunda iteración, el módulo de geo-localización debe obtener las coordenadas de latitud y longitud.

Dependiendo del proveedor, si es de red o vía GPS, se instancia a *getLastKnownLocation* que permite extraer las coordenadas de latitud y longitud. Se agrega estas líneas al código existente⁷.

```
if (locationManager != null) {
    location = locationManager
        .getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
    if (location != null) {
        latitude = location.getLatitude();
        longitude = location.getLongitude();
    }
}
```

Figura 31. Módulo de Geo-localización - Obtener coordenadas vía red.
Fuente: recuperado de <http://goo.gl/9PIaU9>

```
if (locationManager != null) {
    location = locationManager
        .getLastKnownLocation(LocationManager.GPS_PROVIDER);
    if (location != null) {
        latitude = location.getLatitude();
        longitude = location.getLongitude();
    }
}
```

Figura 32. Módulo de Geo-localización - Obtener coordenadas vía GPS.
Fuente: recuperado de <http://goo.gl/9PIaU9>

Una vez almacenadas las coordenadas, se puede instanciar esta clase para ser invocada en conjunto con el módulo de lectura OBD-II para el envío de datos hacia el módulo de gestión de datos.

3.3. Módulo de lectura OBD-II

Las APIs necesarias para la comunicación entre el scanner OBD-II y la aplicación están contenidas en el paquete *android.bluetooth*, la clase *BluetoothDevice* es usada para dispositivos remotos y puede crear un *BluetoothSocket* que es el método encargado de obtener información del scanner Bluetooth.

El *BluetoothSocket* permite conectar dos dispositivos Bluetooth con un socket para intercambiar datos. Para esto, uno de los dispositivos debe iniciar la clase *BluetoothServerSocket* y esperar a recibir peticiones, retorna un *BluetoothSocket* para conectarse.

Para que dos dispositivos se puedan conectar, ambos deben tener una clase *BluetoothSocket* conectadas en el mismo canal RFCOMM (Radio Frequency COMMunication).

⁷ Ver Módulo de Geo-localización. Iteración 1. Clase geo.java

En primer lugar tenemos que conseguir un *BluetoothSocket* llamando al método *createRfcommSocketToServiceRecord(UUID)*. El UUID (Identificador único universal) es una cadena de 128 bits que se utiliza para identificar de forma exclusiva el servicio Bluetooth. Una vez creado el socket, se inicia la conexión con *connect()*, donde el dispositivo remoto asegura que el UUID es correcto y acepta la conexión.

Por último, una vez que los dos dispositivos están conectados, se tiene que gestionar la conexión de datos y de lectura-escritura a través de la conexión Bluetooth. Utilizando el *BluetoothSocket*, se gestiona la entrada y salida de datos a través de *getInputStream()* y *getOutputStream()*. Se envía y se recibe datos usando *read(byte[])* y *write(byte[])*. También es importante cerrar la conexión una vez se cierra el programa, utilizando el comando *cancel()* del *BluetoothSocket*.

3.3.1. Iteración 1.

Para esta primera iteración, este módulo debe gestionar las conexiones Bluetooth y también se deben establecer los comandos OBD a enviar.

3.3.1.1. Clase bluetooth.java

Esta clase se compone de los métodos para identificar la interfaz Bluetooth en el dispositivo, obtener la lista de dispositivos Bluetooth vinculados y establecer una conexión con el scanner OBD-II, cuyo nombre para encontrarlo siempre es "OBDII" ya que de fábrica viene con dicho nombre.

```
//Obtener la interfaz bluetooth del dispositivo
final BluetoothAdapter interfazBluetooth = BluetoothAdapter.getDefaultAdapter();
if (interfazBluetooth == null) {
    //El dispositivo no cuenta con una interfaz bluetooth.
    return sin_bluetooth;
} else if (interfazBluetooth.isEnabled()) {
    if (hayDispVinculados()) {
        //Si la interfaz bluetooth está activada, obtiene la lista (set) de los dispositivos
        Set<BluetoothDevice> setDispositivos = interfazBluetooth.getBondedDevices();
        //Inicia la búsqueda de los dispositivos OBD en una lista (set).
        Set<BluetoothDevice> dispOBDS = new HashSet<BluetoothDevice>();
        for (BluetoothDevice device : setDispositivos) {
            if (cancelar)
                break;
            if (device.getName().equals("OBDII")) // Solo dispositivos llamados OBDII
                dispOBDS.add(device);
        }
    }
}
```

Figura 33. Módulo de Lectura OBDII - Interfaz Bluetooth y búsqueda.
Fuente: el autor.

Como establece en la documentación oficial, se debe crear un socket de comunicación de dos vías (*inputstream*, *outputstream*) para poder enviar y recibir información, se necesitará un UUID (Universal Unique Identifier)(Google 2016a) que servirá para comunicarse con el scanner OBD-II.

```
BluetoothSocket sock = null;
try {
    for (BluetoothDevice dev : listOfBDs) {
        if (cancelar) {
            break;
        }
        Logger.i("BT", "Intentando conectarse a " + dev.getAddress());
        //Intenta y se conecta con el primer dispositivo BT que encuentre
        //Tipo de conexión Bluetooth serial: compatible con dispositivos Bluetooth 2.1 en adelante
        //Codigo de Android Developers API: http://developer.android.com/intl/es/reference/android/bluetooth/BluetoothDevice
        BluetoothSocket socket = dev.createBluetoothSocketToServiceRecord(UUID.fromString("00001101-0000-1000-8000-000027383838"));
        socket.connect();
        sock = socket;
        if (sock.isConnected()) {
            Logger.i("BT", "Conexión establecida: " + dev.getAddress());
        } else {
            Logger.w("BT", "Conexión fallida: " + dev.getAddress());
        }
        break;
    }
}
//Stream usado para la lectura de bytes en la transmisión de información.
//Referencia: http://developer.android.com/intl/es/reference/java/io/InputStream.html
InputStream in = null;
OutputStream out = null;
//Comenzamos el canal socket: sigue leyendo...
if (sock == null) {
    throw new IOException("Falló conectándose al dispositivo OBD");
}
//Transmisión de bytes
in = sock.getInputStream();
out = sock.getOutputStream();
```

Figura 34. Módulo de Lectura OBDII - Conexión Bluetooth con un dispositivo.
Fuente: el autor.

3.3.1.2. Clase obd.java

Esta clase establece los comandos OBD a usar. Por defecto, OBD-II trabaja con multitud de comandos, sin embargo se establecen nueve comandos que son considerados principales para dar un diagnóstico general del estado del vehículo⁸.

Los comandos a usar son los siguientes:

Tabla 4. Comandos OBDII a usar.

Comando	Descripción	Función	Rango
0104	Carga del Motor	Determinar el esfuerzo del motor en un determinado instante	0 – 100 %
0105	Temperatura del Refrigerante	La temperatura del líquido refrigerante usado para enfriar el motor	75 – 95 °C
010C	RPM	Revoluciones por minuto del motor	0 - 16,383 rpm
010D	Velocidad	Velocidad de marcha del vehículo	0 – 255 km/h
010A	Presión de Combustible	La presión interna del compartimiento de combustible	0 – 765 kPa
0110	MAF	Masa de flujo de aire, importante para el cálculo de consumo de combustible	0 – 655 gr/sec
0146	Temperatura Ambiente	Temperatura ambiental	-40 – 215 °C
0151	Tipo de Combustible	Tipo de combustible usado, como etanol, gasolina, diésel, etc.	
015E	Tasa de Combustible del motor	Cantidad de combustible entrante en el motor	0 – 3212 L/h

Nota Fuente: OBDII PIDS Wikipedia.

⁸ Consideración en base al criterio del autor.

En base a esto, se puede definir la clase OBD de la siguiente forma:

```
public enum EnumComandos {  
  
    //Lista de comandos a ejecutarse.  
    //Lista completa de Wikipedia http://en.wikipedia.org/wiki/OBD-II\_PIDs  
    Cod0104("0104", "Carga del motor"),  
    Cod0105("0105", "Temperatura del refrigerante"),  
    Cod010C("010C", "RPM"),  
    Cod010D("010D", "Velocidad"),  
    Cod010A("010A", "Presión del combustible"),  
    Cod0110("0110", "Masa de flujo de aire MAF"),  
    Cod0146("0146", "Temperatura ambiente"),  
    Cod0151("0151", "Tipo de combustible"),  
    Cod015E("015E", "Tasa de combustible del motor");  
}
```

Figura 35. Módulo de Lectura OBDII - Comandos OBDII.
Fuente: el autor.

3.3.2. Iteración 2.

Para esta segunda iteración, este módulo debe enviar los comandos OBD establecidos en la primera iteración a través del socket de conexión con el scanner Bluetooth.

3.3.2.1. Clase obd.java

Se agrega el código para ejecutar los comandos OBD enumerados anteriormente⁹ y también para recepción de la respuesta.

```
public void setInputStream(InputStream in) {  
    this.in = in;  
}  
  
public void setOutputStream(OutputStream out) {  
    this.out = out;  
}  
  
public void run() {  
    enviacmd(cmd);  
    leeResult();  
}  
  
// Envía el comando en forma de bytes  
protected void enviacmd(String cmd) {  
    try {  
        cmd += "\r\n";  
        out.write(cmd.getBytes());  
        out.flush();  
    } catch (Exception e) {  
    }  
}  
  
//Obtiene el resultado eliminando caracteres adicionales  
protected void leeResult() {  
    byte c = 0;  
    this.buff.clear();  
    try {  
        while ((char) (c = (byte) in.read()) != '\r') {  
            buff.add(c);  
        }  
    } catch (IOException e) {  
    }  
}
```

Figura 36. Módulo de Lectura OBDII – Métodos de envío de comandos OBDII.

⁹ Ver Módulo de Lectura OBDII. Iteración 1.

Fuente: el autor.

Se declara dos vías de comunicación del socket que son *setInputStream* y *setOutputStream* que servirán para el envío y recepción del flujo de datos. Los comandos son codificados en una secuencia de bytes.

Para la obtención del resultado, el método *leeResult()* obtiene los bytes recibidos y se eliminan los caracteres ">" que marca el fin de una respuesta obtenida desde el scanner OBD-II.

Definidos estos métodos, procedemos al envío secuencial de comandos:

```
InputStream in = null;
OutputStream out = null;
//Comprobamos el canal (socket) tiene conexión.
if (sock == null)
    throw new IOException("Falla conectándose al dispositivo OBD");
//Transmisión de bytes
in = sock.getInputStream();
out = sock.getOutputStream();

String result = null;
//Iniciamos la comunicación enviando el primer comando OBD de conexión
// ate0 = elimina las respuestas de bytes extendidas (echoOff), es decir sólo devuelve
while ((result == null || !result.contains("OK")) && !cancelar) {
    Obd echoOff = new Obd("ate0");
    result = enviaComando(echoOff, in, out).replace(" ", "");
}
i = 0;
//Inicializa la carpeta y archivos donde se guardarán los datos.
GestionDatos.prepararArchivos(activity);

//Obtiene la lista de comandos a ejecutar.
for (EnumComandos c : EnumComandos.values()) {
    if (cancelar)
        break;
    //Envía el comando a ejecutarse.
    result = enviaComando(new Obd(c.obtCod()), in, out);
}
```

Figura 37. Módulo de Lectura OBDII - Envío secuencial de comandos OBDII.
Fuente: el autor.

Dentro de un ciclo repetitivo, se coloca el envío del comando OBD "ate0" que lo que hace es deshabilitar el tiempo de respuesta estándar, haciendo que los tiempos de espera de respuesta entre el vehículo y el scanner se acoplen al tiempo que tarda en responder el vehículo, esto se debe a que no todos los vehículos responden a un comando con la misma rapidez.

Dentro de otro ciclo repetitivo, se obtiene la lista de los comandos enumerados anteriormente¹⁰ y son pasados como parámetros al método de envío para que se ejecuten uno a uno.

¹⁰ Ver Módulo de Lectura OBDII. Iteración 1.


```

result = enviaComando(new Cmd(c.obcCod()), in, out);
//Si la respuesta contiene bytes se procede a gestionar los datos.
// NODATA = respuesta del sensor cuando no está disponible para lectura de datos.
if (!result.equals("NODATA") || !result.equals("SEARCHING...")) {
//Se comprueba la conexión y se envía al servidor en caso de conexión exitosa o se escribe en
if (hayConexion == true) {
    GestionDatos.subirDatos(c, result, latitud, longitud, identificador, server, fecha, tiempo);
} else {
    GestionDatos.guardarDatos(c, result, latitud, longitud, identificador, fecha, tiempo);
}
}
}

```

Figura 38. Módulo de Lectura OBDII - Recepción de datos.
Fuente: el autor.

Una vez ejecutado un comando, pueden existir tres opciones de respuesta:

- Respuesta “NODATA”: El comando no es soportado por el sensor.
- Respuesta “SEARCHING...”: Existe incompatibilidad entre el scanner OBDII y el vehículo.
- Respuesta hexadecimal: Si se recibe un dato hexadecimal, el sensor ha transmitido su respuesta de forma satisfactoria.

Por último, se verifica la conectividad del dispositivo para según ello, enviar los datos al módulo de gestión de datos.

3.4. Módulo de gestión de datos

3.4.1. Iteración 1.

En esta iteración, el módulo debe determinar la conectividad del dispositivo. Para eso, se hace uso de la instancia *ConnectivityManager* que accede al servicio de conectividad del dispositivo.

3.4.1.1. Clase *testConectividad.java*

Se define la clase invocando a *ConnectivityManager*, este determina por medio del método *getActiveNetworkInfo* si existe alguna conexión de red activa en el dispositivo.

```

public class TestConectividad {
//Determinar si hay conexión internet via Wifi/3G/4G.
//Código tomado de StackOverflow: http://stackoverflow.com/questions/423801/detect-whether-there-is
private static TestConectividad instance = new TestConectividad();
static Context context;
ConnectivityManager connectivityManager;
NetworkInfo wifiInfo, mobileInfo;
boolean connected = false;

public static TestConectividad getInstance(Context ctx) {
    context = ctx.getApplicationContext();
    return instance;
}

public boolean isOnline() {
    try {
        connectivityManager = (ConnectivityManager) context
            .getSystemService(Context.CONNECTIVITY_SERVICE);

        NetworkInfo networkInfo = connectivityManager.getActiveNetworkInfo();
        connected = networkInfo != null && networkInfo.isAvailable() && networkInfo.isConnected();
        return connected;
    } catch (Exception e) {
        System.out.println("CheckConnectivity Exception: " + e.getMessage());
        Log.v("connectivity", e.toString());
    }
    return connected;
}
}

```

Figura 39. Módulo de Gestión de Datos - Clase para determinar la conectividad.
Fuente: recuperado de <http://goo.gl/RIAT8f>

3.4.2. Iteración 2.

Para esta segunda iteración, este módulo gestionar los datos en función de la conectividad del dispositivo.

3.4.2.1. Clase gestionDatos.java

Para la gestión de datos, se tienen dos enfoques: guardar los datos en un archivo CSV en el almacenamiento interno del dispositivo móvil, o enviar los datos directamente al servidor para su almacenamiento en el servidor de datos en la nube.

3.4.2.1.1. Almacenamiento en el dispositivo

Para el almacenamiento interno, primero se debe obtener la ruta en dónde se va a crear el directorio, para lo cual se crea una carpeta llamada "OBD2". De la misma forma, en la misma ruta se crea el archivo "datos" de extensión CSV.

```
//Crea la carpeta OBD2 en el almacenamiento interno del dispositivo.
public static final String DIRECTORIO = Environment.getExternalStorageDirectory() + "/OBD2/";
//Crea el archivo con nombre y extensión.
public static final String DATOS = DIRECTORIO + "datos.csv";
```

Figura 40. Módulo de Gestión de Datos - Ruta de almacenamiento de datos.
Fuente: el autor.

Una vez definidas las rutas del directorio y del archivo, se procede a crear dicho sistema de archivo en el almacenamiento interno del dispositivo.

```
public static void prepararArchivos(Context context) {
    //Crea el directorio en la ubicación especificada al inicio.
    new File(DIRECTORIO).mkdirs();
    File archOBD = new File(DATOS);
    if (archOBD.exists()) {
        //archOBD.delete();
    } else {
        //Crea el archivo CSV para los datos.
        try {
            archOBD.createNewFile();
            BufferedWriter buf = new BufferedWriter(new FileWriter(archOBD, true));
            buf.append("FECHA, TIEMPO, IDENTIFICADOR, LATITUD, LONGITUD, DESCRIPCION, COMANDO, DATO");
            buf.newLine();
            buf.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Figura 41. Módulo de Gestión de Datos - Creación de archivos en el dispositivo.
Fuente: el autor.

Primero se crea el directorio, dependiendo de la existencia del archivo CSV se inicializan las cabeceras de datos.

Si el archivo está inicializado, se pasa a la inserción de una fila de datos.

```

//Retorna para escribir los datos en formato csv.
public static void guardarDatos(EnumComando c, String result, Double lati, Double longi, String identifi, String fecha,
//Variables
latitud = lati;
longitud = longi;
identificador = identifi;
fecha = fecha;
tiempo = tiempo;

//Ubica el archivo y el directorio
new File(DIRECTORIO).mkdir();
File archivo = new File(DATASND);
try {
//Escribe el archivo asignando un nuevo file con los datos.
BufferedWriter buf = new BufferedWriter(new FileWriter(archivo, true));
buf.append(fecha + "," + tiempo + "," + identificador + "," + latitud + "," + longitud + "," + c.obtDescripcion() +
buf.newLine();
buf.close();
} catch (IOException e) {
e.printStackTrace();
}
}

```

Figura 42. Módulo de Gestión de Datos - Inserción de datos en el CSV.
Fuente: el autor.

El método de *guardarDatos()* recibe los parámetros provenientes del módulo de Lectura OBD-II, se busca la ruta mediante el directorio y el archivo CSV. Una vez localizados, se abre el archivo para edición.

Se ingresa una línea con los datos recibidos y finaliza la edición agregando un salto de línea para el siguiente registro. Con ello, el almacenamiento de datos en el dispositivo queda completo.

3.4.2.1.2. Transmisión al servidor

Para transmitir los datos se hace uso de los pares *NameValuePair*, que son una lista de parámetros “llave, valor” usados frecuentemente para representar los parámetros de una petición *http*.

Para esto, el método para subir los datos al servidor debe contener un parámetro con la URL del servidor.

```

try {
List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);
nameValuePairs.add(new BasicNameValuePair("fecha", fecha));
nameValuePairs.add(new BasicNameValuePair("tiempo", tiempo));
nameValuePairs.add(new BasicNameValuePair("latitud", String.valueOf(latitud)));
nameValuePairs.add(new BasicNameValuePair("longitud", String.valueOf(longitud)));
nameValuePairs.add(new BasicNameValuePair("descripcion", c.obtDescripcion()));
nameValuePairs.add(new BasicNameValuePair("comando", c.obtCod()));
nameValuePairs.add(new BasicNameValuePair("datos", result));
nameValuePairs.add(new BasicNameValuePair("identificador", identificador));
nameValuePairs.add(new BasicNameValuePair("add", "true"));
HttpPost httpPost = new HttpEntity(nameValuePairs);
HttpResponse response = httpClient.execute(httpPost);
HttpEntity resEntity = response.getEntity();

if (resEntity != null) {
String responseStr = EntityUtils.toString(resEntity).trim();
}

} catch (IOException e) {
}
}

```

Figura 43. Módulo de Gestión de Datos - Subida de datos al servidor.
Fuente: el autor.

Se crea la lista de pares para lo cual se agregan cada uno de los respectivos identificadores con sus datos.

Se envía la petición al servidor por medio de *httppost* y se recibe la respuesta dentro del objeto *resEntity*. Con la subida satisfactoria, el módulo de gestión de datos queda completo.

3.5. Módulo de configuración

3.5.1. Iteración 1.

Este módulo establece la configuración de la aplicación de acuerdo a los parámetros establecidos por el usuario, leyendo desde el archivo de configuración de la aplicación ubicado en el almacenamiento interno del dispositivo. El archivo lleva por nombre “obdconfig” y tiene una extensión .txt con la siguiente estructura:

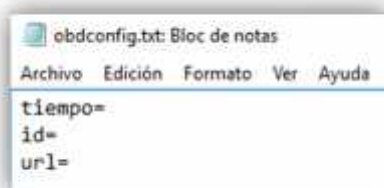


Figura 44. Módulo de Configuración - Estructura del archivo de configuración.
Fuente: el autor.

Los parámetros a especificar son:

- Tiempo: el tiempo entre lecturas de datos desde el vehículo, expresado en segundos, aunque la aplicación recibirá este tiempo transformado en unidad de mili-segundos.
- ID: El identificador que será usado para acceder a la visualización de datos en la aplicación web.
- URL: La dirección del servidor hacia donde se envían los datos.

La configuración por defecto, si no se ha especificado otros valores, es la siguiente:

- tiempo=180
- id=test
- url=http://tesisobd.herokuapp.com/procesar.php

3.5.1.1. Clase leerConfig.java

La clase inicia realizando la búsqueda del archivo en el dispositivo.

```

public static void obtenerParametros(){
    try {
        //Lectura del archivo de configuración
        //Código tomado de StackOverflow: http://stackoverflow.com/questions/10545741/how
        FileInputStream is;
        BufferedReader reader;
        File dir = Environment.getExternalStorageDirectory();
        File file = new File(dir,"obdconfig.txt");

        if (file.exists()) {
            is = new FileInputStream(file);
            reader = new BufferedReader(new InputStreamReader(is));
            String line = reader.readLine();
            int i = 0;
            while (line != null) {
                parametros[i] = line;
                line = reader.readLine();
                i++;
            }
        }
    } catch (IOException e) {
        Toast.makeText(actividad,"Error al abrir el archivo!", Toast.LENGTH_LONG).show();
    }
}
}

```

Figura 45. Módulo de configuración - Lectura del archivo de configuración.
Fuente: el autor.

Primero, se obtiene la ruta del almacenamiento interno (sdcard). Se especifica el nombre del archivo a buscar.

Luego se hace la comparación lógica si el archivo existe, se invoca los métodos de java para lectura de datos. Dentro de un ciclo repetitivo, se almacena un arreglo con las líneas obtenidas del archivo.

Como en la lectura realizada se guarda toda la línea, es necesario eliminar los caracteres innecesarios para luego obtener el parámetro real, para lo cual cada parámetro tiene su propio método, el cual establecerá la configuración de cada uno con el valor real.

```

//Obtiene el tiempo en segundos especificado en el archivo
public static int obtenerTiempo(){
    String aux, tmp;
    int num;
    aux = parametros[0];
    //elimina el texto innecesario.
    tmp = aux.replace("tiempo=", "");
    num = Integer.valueOf(tmp);
    //transformación a mili-segundos
    tiempo = num * 1000;
    return tiempo;
}

```

Figura 46. Módulo de Configuración - Obtener parámetro de tiempo.
Fuente: el autor.

El método *obtenerTiempo()* toma el primer valor del arreglo, que es la línea que contiene el tiempo de actualización de lectura de datos. Se eliminan los caracteres innecesarios, el valor restante al ser de tipo string, tiene que convertirse a entero y por

último a mili-segundos. Como resultado se retorna la variable *tiempo* que contiene el parámetro configurado hacia la clase de Lectura OBD-II.

Así mismo, se establece el método *obtenerId()* que, similar al método anterior, lee el valor configurado para el identificador.

```
//Obtiene el identificador para posteriormente consultar los datos.  
//el id puede ser la placa del vehiculo o cualquier valor.  
public static String obtenerId(){  
    String aux, tmp;  
    aux = parametros[1];  
    tmp = aux.replace("id=", "");  
    id = tmp;  
    return id;  
}
```

Figura 47. Módulo de Configuración - Obtener parámetro identificador.
Fuente: el autor.

Se realiza la lectura del arreglo en la posición superior al método anterior, se elimina el texto adicional y se asigna la variable para su retorno.

```
//Obtiene la dirección del servidor a donde se desea enviar los datos.  
public static String obtenerUrl(){  
    String aux, tmp;  
    aux = parametros[2];  
    tmp = aux.replace("url=", "");  
    url = tmp;  
    return url;  
}
```

Figura 48. Módulo de Configuración - Obtener parámetro servidor.
Fuente: el autor.

Por último el método *obtenerUrl()* realiza la misma lógica para obtener el valor de url del servidor hacia donde se envían los datos.

Al término de esta iteración se tiene lista la aplicación móvil para pruebas.

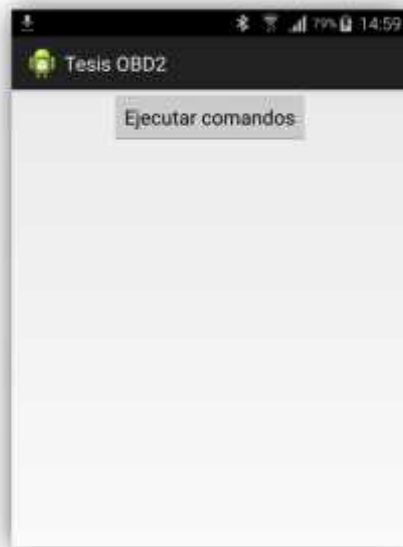


Figura 49. Interfaz de la aplicación móvil.
Fuente: el autor.

3.6. Módulo web de carga de datos

3.6.1. Iteración 1.

En esta iteración, el módulo web de carga de datos debe definir el modelo de base de datos a usar.

Teniendo en cuenta la simplicidad de la estructura de la información a recoger, se establece el modelo de datos contenido en una sola tabla, de la siguiente forma:

Datos		
<u>id</u>	int	<pk>
fecha	varchar(250)	
tiempo	varchar(250)	
descripcion	varchar(500)	
comando	varchar(250)	
latitud	varchar(250)	
longitud	varchar(250)	
dato	varchar(250)	
identificador	varchar(250)	

Figura 50. Módulo Web de Carga de Datos - Base de datos.
Fuente: el autor.

El script de base de datos es generado por la herramienta de modelado Power Designer, de la siguiente forma:

```

drop table if exists DATOS;

/* Tabla: DATOS */
create table DATOS
(
  ID          int not null,
  FECHA       varchar(250),
  TIEMPO      varchar(250),
  DESCRIPCION varchar(500),
  COMANDO     varchar(250),
  LATITUD     varchar(250),
  LONGITUD    varchar(250),
  DATO        varchar(250),
  IDENTIFICADOR varchar(250),
  primary key (ID)
);

```

Figura 51. Módulo Web de Carga de Datos - Script de base de datos.
Fuente: el autor.

Generado este script con extensión .sql podemos importarlo al servidor de base de datos ClearDB usando el siguiente comando batch:

```

mysql --host=us-cdbr-east.cleardb.com --user=USR --password=xxxxx --reconnect
heroku_xxxxxx < base.sql

```

Una vez completada esta acción, el servidor de datos está preparado para su funcionamiento, resta crear completar la lógica de inserción y consulta de datos.

3.6.2. Iteración 2.

El módulo web de carga de datos en esta iteración tiene que completar lógica de subida de datos desde la aplicación móvil y el formulario de subida de archivo en la aplicación web.

3.6.2.1. Interfaz web de subida de archivo CSV

La interfaz web consta de un formulario donde se subirá un archivo CSV que es generado desde la aplicación.

```

<div class="col-lg-4">
  <div>Subir archivo</div>
  <div style="text-align: justify;">Suba el archivo .csv generado desde la aplicación.
  El archivo "datos.csv" generado se encuentra en el almacenamiento interno de su dispositivo, en una carpeta llamada "0602".</div>
  <form action="cargadatos.php" method="post" enctype="multipart/form-data">
    <div class="form-group">
      <input class="form-control" type="text" value="Seleccionar archivo:"/>
    </div>
    <div class="text-center">
      <input class="form-control" type="file" name="archivocsv" id="archivocsv" style="align:center;" accept=".csv" required/>
    </div>
    <div class="text-center">
      <input type="submit" class="btn btn-lg btn-success" value="Subir Datos"/>
    </div>
  </form>
</div>

```

Figura 52. Módulo Web de Carga de Datos - Formulario de subida de archivo CSV.
Fuente: el autor.

Dentro de la etiqueta *input* se especifica el parámetro "accept = '.csv'" que limita al navegador a solo abrir archivos con esta extensión.



Figura 53. Módulo Web de Carga de Datos - Interfaz de subida de archivo CSV.
Fuente: el autor.

El formulario envía el archivo a “*cargardatos.php*” el cual maneja la lógica de la subida de archivos.

PHP por medio de la función *fgetcsv()* realiza la lectura del archivo de forma secuencial, almacenando en un arreglo asociativo los valores obtenidos.

```
extract ($_POST);
$row = 0;
$col = 0;

if (isset($_FILES['archivocsv'])) {
    $file = $_FILES['archivocsv']['tmp_name'];
    $handle = fopen($file, "r");

    while (($row = fgetcsv($handle, 4096)) !== false)
    {
        if (empty($fields))
        {
            $fields = $row;
            continue;
        }

        foreach ($row as $k=>$value)
        {
            $results[$col][$fields[$k]] = $value;
        }
        $col++;
        unset($row);
    }
    if (!feof($handle))
    {
        echo "Error: unexpected fgets()";
    }
    fclose($handle);
}
```

Figura 54. Módulo Web de Carga de Datos - Lectura de archivo CSV.
Fuente: el autor.

Realizado este proceso, se procede a establecer la conexión con la base de datos. Antes de realizar la respectiva consulta, se deben convertir los valores hexadecimales de los datos obtenidos a un valor legible.

```
try {
    $db = new PDO($dsn, $db['user'], $db['pass']);
    $db->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    foreach ($results as $row)
    {
        switch ($row['COMANDO']) {
            // Carga del motor (%)
            case "0104":
                $step = substr($row['DATO'], 4);
                $aux = hexdec($step);
                $var = (int)$aux;
                $row['DATO'] = strval(round(($var / 255) * 100));
                break;
            // Temperatura del refrigerante (°C)
            case "0105":
                $step = substr($row['DATO'], 4);
                $aux = hexdec($step);
                $var = (int)$aux;
                $row['DATO'] = strval($var - 40);
                break;
            // Revoluciones del motor (rpm)
            case "010C":
                $step = substr($row['DATO'], 4);
                $aux = hexdec($step);
                $var = (int)$aux;
                $row['DATO'] = strval(round($var / 4));
                break;
        }
    }
}
```

Figura 55. Módulo Web de Carga de Datos - Conversión de datos hexadecimales.
Fuente: el autor.

Dentro de un ciclo repetitivo, se implementa una función *switch* del arreglo asociativo, para poder revisar un valor de una determinada columna, en este caso “\$row[‘comando’]” y poder determinar el tipo de comando para poder convertir su valor en base a su respectiva fórmula¹¹.

El proceso que se sigue es el siguiente:

- Dependiendo del comando, primero se eliminan los 4 primeros caracteres hexadecimales con la función *substr()* para dejar solamente los hexadecimales restantes que contienen el dato a convertir.
- Se emplea la función *hexdec()* que permite convertir números hexadecimales a su equivalente decimal.
- Se asigna a la columna “\$row[‘dato’]” el valor calculado de la fórmula, según el comando OBD(Abrigo Maldonado 2007).

¹¹ Ver Módulo de gestión de datos – Iteración 2.

```

IF($row['LATITUD'] != "0" && $row['LONGITUD'] != "0"){
    $prepare = $db->prepare("INSERT INTO datos
        (id, fecha, tiempo, identificador, latitud, longitud, descripción, comando, dato)
        VALUES('', '$row['FECHA']','','$row['TIEMPO']','','$row['IDENTIFICADOR']','','$row
    $prepare->execute();
}

```

Figura 56. Módulo Web de Carga de Datos - Inserción de datos.
Fuente: el autor.

Una vez realizada la conversión, se hace la consulta de datos en donde se insertan los datos en la base. Es necesario insertar las columnas que tengan una ubicación distinta de cero en las coordenadas de latitud y longitud, para evitar errores en el módulo de visualización de datos, por lo que se hace una comprobación adicional antes de insertar los datos.

3.6.2.2. Subida de datos desde la aplicación

Similar a la carga de datos anterior, los datos que se envían desde la aplicación hacia el servidor deben convertirse de la misma forma antes de ser almacenados, para ello el proceso es el siguiente:

- Se extrae los valores recibidos desde la aplicación¹² y los asigna en variables para un mejor manejo.
- Se implementa la función *switch()* de la variable comando, para determinar que fórmulas usar según el comando.
- Dependiendo del comando, primero se eliminan los 4 primeros caracteres hexadecimales con la función *substr()* para dejar solamente los hexadecimales restantes que contienen el dato a convertir.
- Se emplea la función *hexdec()* que permite convertir números hexadecimales a su equivalente decimal.
- Se asigna a la variable "\$valor" el valor calculado de la fórmula, según el comando OBD(Abrigo Maldonado 2007).

¹² Ver Módulo de gestión de datos – Iteración 2

```

require 'config/basedatos.php';

// Obtener valores
$fecha = $_POST["fecha"];
$tiempo = $_POST["tiempo"];
$lati = $_POST["latitud"];
$longi = $_POST["longitud"];
$descrip = $_POST["descripcion"];
$comando = $_POST["comando"];
$valor = $_POST["dato"];
$identificador = $_POST["identificador"];

// Procesar la data

switch ($comando) {
    // Carga del motor (%)
    case "0104":
        $tmp = substr($valor, 4);
        $aux = hexdec($tmp);
        $var = (int)$aux;
        $valor = strval(round(($var / 255) * 100));
        break;
    // Temperatura del refrigerante (°C)
    case "0105":
        $tmp = substr($valor, 4);
        $aux = hexdec($tmp);
        $var = (int)$aux;
        $valor = strval($var - 40);
        break;
}

```

Figura 57. Módulo Web de Carga de Datos - Conversión de hexadecimal desde la aplicación móvil.
Fuente: el autor.

Realizada la conversión, se prepara la consulta para inserción de datos en la base. Se establece la conexión con algunos parámetros para captura de excepciones.

Ya que este no es un proceso iterativo, cada vez que la aplicación envíe datos, el servidor realizará estas operaciones.

```

try {
    $db = new PDO($dsn, $db['user'], $db['pass']);
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $prepare = $db->prepare("INSERT INTO datos
        (id, fecha, tiempo, descripcion, comando, latitud, longitud, dato, identificador)
        VALUES ('', '$fecha', '$tiempo', '$descrip', '$comando', '$lati', '$longi', '$valor', '$identificador')");
    $prepare->execute();
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}

```

Figura 58. Módulo Web de Carga de Datos - Inserción de datos enviados desde la aplicación móvil.
Fuente: el autor.

Con este módulo completo, resta la exportación de datos para descarga y el trazo de rutas entre ubicaciones dentro del mapa de Google.

3.7. Módulo web de visualización de datos

3.7.1. Iteración 1.

Este módulo se encarga de generar la visualización de los recorridos realizados, a través del consumo del API de Google Maps.

Para realizar la visualización de recorridos, es necesario obtener los rangos de fecha y hora, los mismos que se construyen bajo el lenguaje HTML usando librerías de Bootstrap para los estilos.

3.7.1.1. Interfaz de consulta

Se genera el archivo "*index.php*" para el diseño de la interfaz de consulta, que inicia por la declaración de librerías, que incluyen a Bootstrap y JQuery:

```
<!DOCTYPE html >
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
<link rel="stylesheet" href="//code.jquery.com/ui/1.11.4/themes/smoothness/jquery-ui.css">
<link rel="stylesheet" href="//getbootstrap.com/examples/jumbotron-narrow/jumbotron-narrow.css">
<link rel="stylesheet" href="//getbootstrap.com/dist/css/bootstrap.min.css">
<link rel="stylesheet" type="text/css" href="css/starter-template.css">
<script src="//code.jquery.com/jquery-1.10.2.js"></script>
<script src="//code.jquery.com/ui/1.11.4/jquery-ui.js"></script>
```

Figura 59. Módulo Web de Visualización - Declaración de librerías.
Fuente: el autor.

El siguiente paso es construir el formulario para obtener los rangos de fechas, se especifican los siguientes parámetros:

- ID: Identificador asignado dentro del archivo de configuración en la aplicación móvil.
- Desde: Inicio del rango de fecha de la consulta.
- Hasta: Fin del rango de fecha de la consulta.
- Hora inicio: Inicio del rango de tiempo de la consulta.
- Hora fin: Fin del rango de tiempo de la consulta

```
<form action="recorridos.php" method="POST" >
<label for="identificador">ID:</label><input type="text" name="identificador" placeholder="Placa del vehículo o Email" required></input>
<label for="from">Desde:</label><input type="text" id="from" name="desde" required></input>
<label for="to">Hasta:</label><input type="text" id="to" name="hasta" required></input>
<label for="horaini">Entre las horas:</label><input type="text" id="horaini" name="horaini">
<select id="horaini" name="horaini">
<option value="01:00:00">01H00</option>
<option value="02:00:00">02H00</option>
<option value="03:00:00">03H00</option>
```

Figura 60. Módulo Web de Visualización - Formulario de consulta.
Fuente: el autor.

Una vez especificados los campos del formulario, se envía la información del formulario hacia el archivo “*recorridos.php*” que contiene la lógica de consultas e interfaz de visualización.



Figura 61. Módulo Web de Visualización - Interfaz de consulta.
Fuente: el autor.

3.7.1.2. Interfaz de Visualización

El archivo “*recorridos.php*” contiene la interfaz de visualización de datos. La declaración de librerías es similar a la interfaz anterior.

```
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<meta http-equiv="content-type" content="text/html; charset=UTF-8" />
<title>recorridos realizados</title>
<script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDh12bWqjW6cF1jyfaWfIDffYMcCfY8" type="text/javascript"></script>
<link rel="stylesheet" type="text/css" href="https://developers.google.com/maps/documentation/javascript/examples/default.css">
<link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
<link rel="stylesheet" type="text/css" href="css/jumbotron-sarraz.css">
<link rel="stylesheet" type="text/css" href="css/starter-template.css">
<script src="js/jquery-2.2.0.min.js"></script>
```

Figura 62. Módulo Web de Visualización – Declaración de API Google Maps.
Fuente: el autor.

Se hace la adición tanto de una llamada de JavaScript y de un estilo que hace referencia al API de Google Maps. Este estilo servirá para poder mostrar el mapa en la página.

Para visualizar el mapa, se necesita generar una llave (key) que debe ser generada en el sitio de desarrolladores de Google¹³.

¹³ Ver Anexo 1. API de Google Maps.

```
<body>
  <div class="container">
    <div class="header clearfix">
      <nav>
        <ul class="nav nav-pills pull-right">
          <li role="presentation"><a href="/">Inicio</a></li>
          <li role="presentation" class="active"><a href="#">Consultas</a></li>
        </ul>
      </nav>
      <h3 class="text-muted">Datos OBD</h3>
    </div>
    <div class="jumbotron">
      <h1 class="text-muted">Resultados</h1><br>
      <div id="map-canvas"></div>
    </div>
  </div>
</body>
```

Figura 63. Módulo Web de Visualización – Interfaz de visualización.
Fuente: el autor.

Google Maps requiere que un elemento HTML tenga como id “map-canvas” que es el lugar en donde hará el render del mapa, por lo que es colocado dentro de la etiqueta *body*.

Una vez codificada la interfaz, se procede a hacer el consumo de las funciones de Google Maps para dibujar cada una de las ubicaciones, previa la elaboración de la consulta de datos.

3.7.1.3. Consumo del API Google Maps

El primer paso es inicializar el mapa de Google, para lo cual se declara una función JavaScript *initialize()* la que consta de:

```
function initialize() {
  var mapaLoja = new google.maps.LatLng(-4.008417, -79.211121);
  var mapOptions = {
    zoom: 12,
    center: mapaLoja,
    mapTypeId: google.maps.MapTypeId.ROADMAP
  }
  var map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);
  var infowindow = new google.maps.InfoWindow();
}
```

Figura 64. Módulo Web de Visualización - Inicialización del mapa de Google.
Fuente: el autor.

- Coordenadas predeterminadas, que determinan el área que aparecerá en el mapa, en este caso se ubican las coordenadas -4.008417, -79.211121 de latitud y longitud respectivamente, las cuales marcan la ciudad de Loja, Ecuador.
- Zoom, que establece el acercamiento predeterminado del área, 12 en este caso, mientras mayor es el valor, cubre una zona en particular.
- Tipo de mapa, que varía entre las opciones de: ruta, tierra, híbrido. En este caso, el tipo es ruta, el cual permite demarcar las calles con nombres.

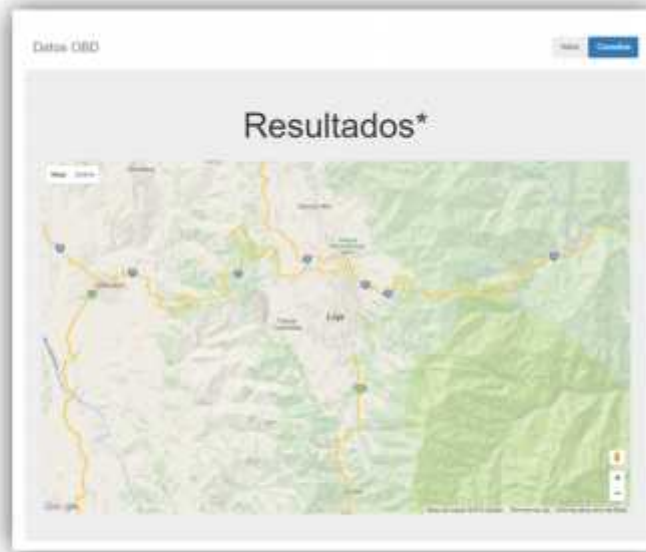


Figura 65. Módulo Web de Visualización - Interfaz de visualización
Fuente: el autor.

El segundo paso es generar la consulta de datos, teniendo como parámetros los valores ingresados en el formulario de consulta. PHP interviene realizando las consultas necesarias, por lo cual el proceso es el siguiente:

```

require 'config/basedatos.php';

extract ($_POST);
$fecha1 = $_POST["desde"];
$fecha2 = $_POST["hasta"];
$identificador = $_POST["identificador"];
$hora1 = $_POST["hora1"];
$hora2 = $_POST["hora2"];

try {

    $db = new PDO($dsn, $db['user'], $db['pass']);
    $db->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sql = "SELECT DISTINCT fecha, tiempo, latitud, longitud, identificador
    FROM datos WHERE identificador = '$identificador' AND CONCAT('fecha', ' ', tiempo) BETWEEN '$fecha1 $hora1'
    AND CONCAT('fecha', ' ', tiempo) BETWEEN '$fecha2 $hora2' ORDER BY fecha ASC";
    $db->execute($sql);
    $datos = $db->fetchAll(PDO::FETCH_ASSOC);
    $coordenadas = array();
}

```

Figura 66. Módulo Web de Visualización - Consulta de datos.
Fuente: el autor.

- Se hace la llamada a un archivo de configuración “*basedatos.php*” que contiene la información de acceso a la base de datos.
- Se extrae los valores obtenidos desde el formulario de consulta, y se almacenan en variables para su manipulación.
- Se hace instancia al método de conexión PDO para establecer la conexión con la base de datos.
- Se establece atributos de conexión que servirán para el manejo de excepciones.

- Se formula la consulta sobre la tabla de datos, estableciendo los rangos de fecha y hora provistos en el formulario, además se ordena el resultado por fecha.
- Se declara un arreglo que almacenará las coordenadas de latitud y longitud, misma que servirá para posteriormente realizar el trazado de rutas entre ubicaciones.

Obtenidos los datos, se procede a hacer uso del API de Google Maps colocando las ubicaciones dentro del mapa, por medio de JavaScript, esto se hace dentro de un ciclo repetitivo mientras la consulta devuelva resultados, por lo tanto, se tiene el siguiente proceso:

```

while ($result = $prepare - fetch($PDO - FETCH_ASSOC)) {
    echo "var titulo = " . $result['fecha'] . " - " . $result['tiempo'] . " ";
    echo "var pois = new google.maps.LatLng( " . $result['latitud'] . " , " . $result['longitud'] . " );";
    echo "var marker = new google.maps.Marker({
        position: pois,
        map: map,
        title: titulo,
        content: titulo
    });";
    echo "google.maps.event.addListener(marker, 'click', function() {
        infowindow.setContent(this.content);
        infowindow.open(map, this);
    });";

    $coordenadas[] = array($result['latitud'], $result['longitud']);
}

```

Figura 67. Módulo Web de Visualización – Impresión de ubicaciones en el mapa.
Fuente: el autor.

- Se declara una variable que contiene la fecha y hora de la ubicación.
- Se declara un nuevo marcador, que será una nueva ubicación señalada dentro del mapa, con los datos de latitud, longitud.
- Se crea el marcador, con la ubicación y el título, en este caso el título está compuesto por la fecha y hora.
- Se llama a la función *infowindow()* de Google la cual agrega la ventana desplegable en un clic con la información en cada marcador.
- Por último se van almacenando las coordenadas para un posterior uso en trazado de rutas.

3.7.2. Iteración 2.

Para esta iteración, el módulo de visualización debe trazar líneas entre marcadores en base a la fecha de cada uno de estos, estableciendo una ruta.

Esto se consigue instanciando una función del API de Google Maps llamada PolyLine, que dibuja una línea de conectando cada marcador en un mapa.

Modificando la interfaz de visualización¹⁴ al final del código para colocar cada marcador en el mapa, podremos usar la función de Polyline usando el arreglo que contiene las coordenadas de latitud y longitud.

¹⁴ Ver Módulo Web de Visualización de Datos. Consumo del API de Google Maps.

```

var polylinePlanCoordinates = [];
var polyline_data = <?php echo json_encode( $coordenadas ); ?>;
for (var i=0;i<polyline_data.length;i++){
    polylinePlanCoordinates.push(new google.maps.LatLng(polyline_data[i][0], polyline_data[i][1]));
}

var path= new google.maps.Polyline({
    path: polylinePlanCoordinates,
    geodesic: true,
    strokeColor: '#FF0000',
    strokeOpacity: 1.0,
    strokeWeight: 2
});

path.setMap(map);

```

Figura 68. Módulo Web de Visualización de Datos - Trazar rutas entre ubicaciones.
Fuente: el autor.

El proceso es el siguiente:

- Se pasa desde PHP hacia JQuery las coordenadas en almacenadas en el arreglo, convirtiendo a formato JSON.
- En un ciclo repetitivo, se agregan al mapa cada una de las líneas, desde una ubicación a otra.
- Se establecen los parámetros de opciones para estas líneas como *geodesic* que establece la ruta corta entre ubicaciones, el color, opacidad y grosor de la línea.
- Una vez configuradas las opciones, la función *path.setMap()* coloca las líneas visibles en el mapa.

Al ser este un proceso iterativo, este procedimiento usado es suficiente para graficar una ruta, independientemente del número de marcadores en el mapa. Queda limitado solo para graficar una sola ruta a la vez.

3.8. Módulo web de exportación de datos

3.8.1. Iteración 1.

El módulo de exportación de datos va asociado con el módulo de Visualización de Datos ya que hace uso de la misma consulta e interfaz.

Se crean botones que direccionan a cada descarga, para ello, cada botón tiene su propio formulario.

```

<div class="starter-template">
  <h2>Descargue Información</h2>
  <p>Descargue los datos consultados en los siguientes formatos: </p>
  <form action="descargajson.php" method="POST" target="_blank">
    <input type="hidden" name="identificador" value="<?php echo $identificador;?>" />
    <input type="hidden" name="desde" value="<?php echo $fecha1;?>" />
    <input type="hidden" name="hasta" value="<?php echo $fecha2;?>" />
    <input type="hidden" name="horaini" value="<?php echo $horaini;?>" />
    <input type="hidden" name="horafin" value="<?php echo $horafin;?>" />
    <input type="submit" class="btn btn-lg btn-success" value="JSON"></form>
  </div>
  <form action="descargacsv.php" method="POST" target="_blank">
    <input type="hidden" name="identificador" value="<?php echo $identificador;?>" />
    <input type="hidden" name="desde" value="<?php echo $fecha1;?>" />
    <input type="hidden" name="hasta" value="<?php echo $fecha2;?>" />
    <input type="hidden" name="horaini" value="<?php echo $horaini;?>" />
    <input type="hidden" name="horafin" value="<?php echo $horafin;?>" />
    <input type="submit" class="btn btn-lg btn-success" value="CSV"></form>
  </div>
</div>

```

Figura 69. Módulo Web de Exportación de Datos - Interfaz de descarga.
Fuente: el autor.

Se envía además los rangos de fechas para la consulta de datos, estos rangos son los mismos que fueron provistos para la visualización de recorridos en el mapa.

Las funciones disponibles en PHP permiten la exportación en varios formatos, se pretende exportar los datos en formatos CSV y JSON.

Primero se crea un script en donde se reciban los datos del formulario, luego de conectar a la base se genera la consulta.

```

extract ($_POST);
$fecha1 = $_POST["desde"];
$fecha2 = $_POST["hasta"];
$identificador = $_POST["identificador"];
$horaini = $_POST["horaini"];
$horafin = $_POST["horafin"];

try {
    $db = new PDO($dsn, $db['user'], $db['pass']);
    $db->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $prepare = $db->prepare("SELECT DISTINCT
    fecha, tiempo, identificador, latitud, longitud, comando, descripcion, dato
    FROM datos WHERE identificador = '$identificador'
    AND CONCAT('fecha', ':', 'tiempo') = '$fecha1 $horaini'
    AND CONCAT('fecha', ':', 'tiempo') = '$fecha2 $horafin'
    ORDER BY fecha ASC");
    $prepare->execute();
    $datos = array();
}

```

Figura 70. Módulo Web de Exportación de Datos - Consulta de datos.
Fuente: el autor.

3.8.1.1. Exportación en formato CSV

Para exportar en formato CSV se crea un script PHP, los resultados que genere la base de datos deben guardarse en un arreglo para poder generar el archivo con el formato adecuado.

El proceso es el siguiente:

- Se declara el arreglo que contendrá los datos.
- Se agregan las cabeceras que serán los identificadores de columna.
- Mientras la base de datos devuelva información, se agrega un nuevo elemento en el arreglo.
- Una vez terminado el proceso de la base de datos, la función *fopen()* se encarga de abrir un archivo.
- Se agregan las funciones *header()* que permiten declarar el tipo de contenido y el nombre de archivo y formato. Esto hará que el navegador interprete la descarga de ficheros.
- Se construye el contenido del archivo CSV con la función *fputcsv()*, generando su descarga automática.

```

$datos = array();
// Columnas
array_push($datos, array("FECHA, TIEMPO, IDENTIFICADOR, LATITUD, LONGITUD, COMANDO_OBD, DESCRIPCION, RESULTADO"));
while ($result = $prepare->fetch(PDO::FETCH_ASSOC)) {
    array_push($datos, array_values($result));
}
// Output array into CSV file
$f = fopen('php://output', 'w');
header('Content-Type: text/csv');
header('Content-Disposition: attachment; filename="datosOBD.csv"');
foreach ($datos as $ferow) {
    fputcsv($f, $ferow);
}

```

Figura 71. Módulo Web de Exportación de Datos - Exportación de archivo CSV.
Fuente: el autor.

3.8.1.2. Exportación en formato JSON

Similar a la exportación anterior, para exportar en formato JSON se crea un script PHP siguiendo el mismo proceso:

- Se declara el arreglo que contendrá los datos.
- Mientras la base de datos devuelva información, se agrega un nuevo elemento en el arreglo.
- Una vez finalizado el retorno de datos desde la base de datos, la función *json_encode()* genera el formato correcto de los datos contenidos en el arreglo.
- Se agregan las funciones *header()* que permiten declarar el tipo de contenido y el nombre de archivo y formato. Esto hará que el navegador interprete la descarga de ficheros.
- La función "echo \$json;" agrega los datos al archivo, generando su descarga inmediata en el navegador.

```

while ($result = $prepare->fetch(PDO::FETCH_ASSOC)) {
    $datos[] = $result;
}
$json = json_encode($datos);
header('Content-disposition: attachment; filename=datosOBD.json');
header('Content-type: application/json');
echo $json;

```

Figura 72. Módulo Web de Exportación de Datos - Exportación de archivo JSON.

Fuente: el autor.

El módulo queda completo, y se agrega a la interfaz de visualización para habilitar las opciones de descarga de archivos.



Figura 73. Módulo Web de Exportación de Datos - Interfaz de exportación de archivos.

Fuente: el autor.

CAPÍTULO 4: PRUEBAS E IMPLEMENTACIÓN

4.1. Elementos de prueba

Para realizar las pruebas primero se consideran los siguientes parámetros:

- Vehículo de prueba: Un vehículo para realizar el test de la aplicación móvil.
- Dispositivo de prueba: dispositivo móvil Android.
- Área de prueba: Limitación de un área de pruebas (ciudad/región).

4.1.1. Vehículo de prueba

Los vehículos desde el año 1996 poseen el puerto OBD-II, generalmente se encuentra situado bajo el volante, junto al panel de circuitos interno. Existen vehículos (algunos modelos Chevrolet) que tienen este conector en la parte central del tablero, bajo la radio. Para las pruebas de la aplicación móvil se seleccionó el siguiente vehículo:



Figura 74. Entrada OBDII del auto Toyota Rav4 2015.
Fuente: el autor.

- Marca: Toyota.
- Modelo: Rav 4.
- Año: 2015.
- Motor: 2.0.
- Tipo: Manual.

4.1.2. Dispositivo de prueba

Para efectuar las pruebas de la aplicación móvil, se seleccionó el siguiente dispositivo Android:



Figura 75. Dispositivo de pruebas Samsung Galaxy Note 4.
Fuente: recuperado de <http://goo.gl/9HxOpU>

- Marca: Samsung
- Modelo: Galaxy Note 4
- Conexiones: Bluetooth 4.1, WiFi doble banda, 4G-LTE categoría 4.
- Procesador: SnapDragon 805 2.7 GHz Quadcore.
- Almacenamiento: 32 GB.
- Ram: 3 GB.
- Sistema: Android 5.1 Lollipop.

4.1.3. Área de prueba

Para la elaboración de pruebas con la aplicación móvil se establecen 2 rutas:

Tabla 5. Áreas de prueba de la aplicación móvil.

TIPO	Ciudad Origen	Ciudad Destino	KM Apróx.
Local	Loja – sector sur	Loja – sector norte	9 km.
Inter-cantonal	Catamayo	Loja	38 km.

Nota Fuente: el autor.

4.2. Pruebas de Validación

Las pruebas de validación se ejecutaron en el vehículo, dispositivo y áreas descritas anteriormente.

Se hizo la recolección de datos con el vehículo alternando la conectividad del dispositivo móvil, es decir, se hicieron pruebas de subida de datos desde la aplicación y también subiendo el archivo CSV generado cuando no hubo conectividad en el dispositivo.

4.2.1. Prueba 1: Recorrido Local

Los parámetros considerados para la prueba de recorrido local son los siguientes:

- **Punto de salida:** Tebaida Alta, sector sur de la ciudad de Loja.

- **Punto de llegada:** Universidad Técnica Particular de Loja, sector norte de la ciudad de Loja.
- **Distancia:** 8-9 km apróx.
- **Conectividad:** Internet móvil HSPDA+ del operador Claro.
- **Fecha de la prueba:** 27-01-2016
- **Hora:** 15h00 a 16H00

4.2.1.1. Aplicación Móvil

Se inicia desde la ubicación sur de la ciudad (Tebaida Alta), la Figura 78 muestra la captura de la aplicación Google Maps del punto de inicio exacto en donde se dio inicio a la prueba.

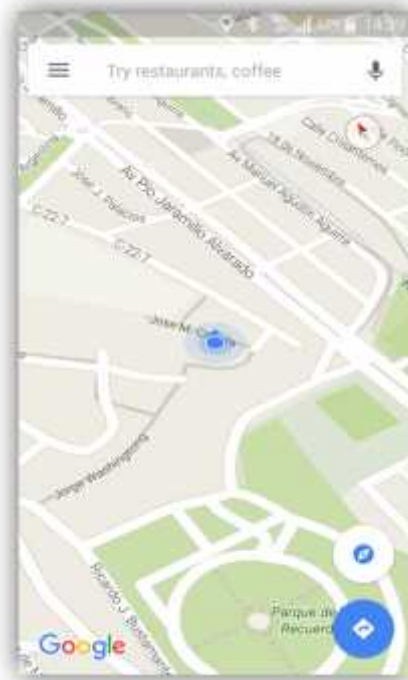


Figura 76. Prueba de validación 1 – Recorrido local: Punto de inicio.

Fuente: el autor.

Se procede a iniciar la aplicación, la Figura 79 muestra la captura de la aplicación cuando comienza a ejecutarse de forma automática, siguiendo el siguiente proceso:

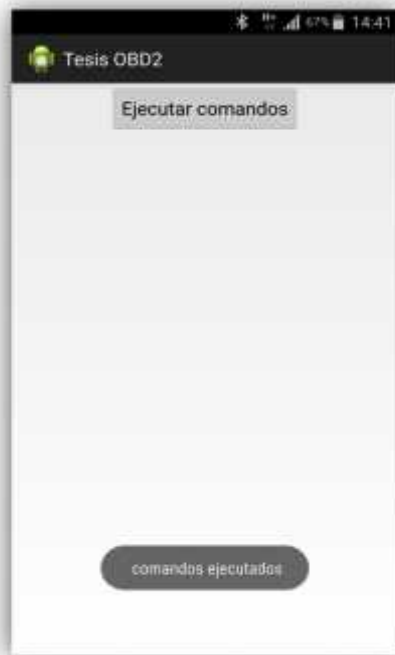


Figura 77. Prueba de validación 1 - Recorrido local: Ejecución de la aplicación.

Fuente: el autor.

- Comprueba los dispositivos Bluetooth.
- Se conecta al scanner OBD-II llamado "OBDII".
- Obtiene la ubicación (geo-localización).
- Envía los comandos OBD.
- Recibe la respuesta.
- Comprueba la conectividad del dispositivo.
 - Si hay conectividad, envía los datos a la aplicación web.
 - Si no hay conectividad, busca el archivo CSV para guardar los datos en el almacenamiento interno del dispositivo.
- Muestra el mensaje de "Comandos Ejecutados".

FECHA	TIEMPO	DESCRIPCION	COMANDO	LATITUD	LONGITUD	DATO	IDENTIFICADOR
2016-01-27	15:13:24	Carga del motor	0104	-4.0158292	-79.2048516	50	kg1234
2016-01-27	15:13:24	Temperatura del ref	0105	-4.0158292	-79.2048516	56	kg1234
2016-01-27	15:13:24	RPM	010C	-4.0158292	-79.2048516	1018	kg1234
2016-01-27	15:13:24	Velocidad	0100	-4.0158292	-79.2048516	10	kg1234
2016-01-27	15:13:24	Masa de flujo de air	0110	-4.0158292	-79.2048516	3	kg1234
2016-01-27	15:13:24	Tipo de combustible	0151	-4.0158292	-79.2048516	Gasolina	kg1234
2016-01-27	15:16:24	Carga del motor	0104	-4.0124433	-79.2033006	82	kg1234
2016-01-27	15:16:24	Temperatura del ref	0105	-4.0124433	-79.2033006	85	kg1234
2016-01-27	15:16:24	RPM	010C	-4.0124433	-79.2033006	1609	kg1234
2016-01-27	15:16:24	Velocidad	0100	-4.0124433	-79.2033006	22	kg1234
2016-01-27	15:16:24	Masa de flujo de air	0110	-4.0124433	-79.2033006	20	kg1234
2016-01-27	15:16:24	Tipo de combustible	0151	-4.0124433	-79.2033006	Gasolina	kg1234
2016-01-27	15:19:24	Carga del motor	0104	-4.0042247	-79.2004418	42	kg1234
2016-01-27	15:19:24	Temperatura del ref	0105	-4.0042247	-79.2004418	56	kg1234
2016-01-27	15:19:24	RPM	010C	-4.0042247	-79.2004418	888	kg1234
2016-01-27	15:19:24	Velocidad	0100	-4.0042247	-79.2004418	18	kg1234
2016-01-27	15:19:24	Masa de flujo de air	0110	-4.0042247	-79.2004418	2	kg1234
2016-01-27	15:19:24	Tipo de combustible	0151	-4.0042247	-79.2004418	Gasolina	kg1234

Figura 78. Prueba de validación 1 – Recorrido local: Datos enviados de la aplicación.

Fuente: el autor.

La Figura 80 muestra los datos almacenados en el servidor, por lo que la aplicación funciona correctamente al momento del envío de datos. Se observa además, que la columna “Dato” muestra los valores en formato legible por lo que la conversión hexadecimal también se realiza correctamente.

Se hizo la prueba de ejecución en segundo plano, como muestra la Figura 81, la aplicación continúa extrayendo datos y ejecutándose normalmente aunque el usuario salga de la interfaz. Se muestra el mensaje de “comandos ejecutados” que implica la correcta ejecución en segundo plano, por lo que el usuario podrá usar otras aplicaciones o bloquear el dispositivo sin problema.



Figura 79. Prueba de validación 1 - Recorrido local: Ejecución en segundo plano.

Fuente: el autor.

La prueba concluye al llegar a la ubicación de destino, la Figura 82 muestra una captura de la aplicación Google Maps del punto exacto de llegada donde finalizó la transmisión de datos.

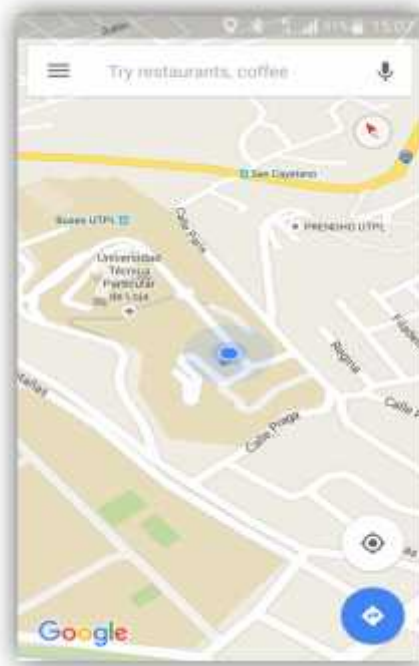


Figura 80. Prueba de validación 1 - Recorrido local: Punto de llegada.

Fuente: el autor.

4.2.1.2. Aplicación Web

Una vez realizada la primera prueba, también es necesario comprobar la correcta visualización de los datos transmitidos durante la misma.

Ingresamos a la aplicación web para realizar la respectiva consulta de datos:

- **Identificador:** lcg1234
- **Fechas desde y hasta:** 27-01-2016
- **Horas:** 15h00 – 16h00

La Figura 83 muestra el resultado devuelto por la aplicación web al realizar la consulta con los datos mencionados.



Figura 81. Prueba de validación 1 - Recorrido local: Resultados en la aplicación web.
Fuente: el autor.

En la Figura 83 se aprecia que los marcadores marcan la ruta desde los puntos de inicio y fin establecidos para la prueba de validación 1. Dando clic en cada marcador, muestra la fecha y hora exacta en la cual fue tomada esa ubicación.

Así mismo, debajo de esta interfaz se encuentran las opciones de descarga de los datos en formato CSV y JSON.

4.2.2. Prueba 2: Recorrido Inter-cantonal

Los parámetros considerados para la prueba de recorrido inter-cantonal son los siguientes:

- **Punto de salida:** San José, sector sur de la ciudad de Catamayo.
- **Punto de llegada:** Tebaida Alta, sector sur de la ciudad de Loja.
- **Distancia:** 38 km apróx.
- **Conectividad:** Sin internet.
- **Fecha de la prueba:** 27-02-2016
- **Hora:** 15h00 a 17H00

4.2.2.1. Aplicación Móvil

Se inicia desde la ubicación sur de la ciudad de Catamayo (San José), la Figura 84 muestra la captura de la aplicación Google Maps del punto de inicio exacto en donde se dio inicio a la prueba.

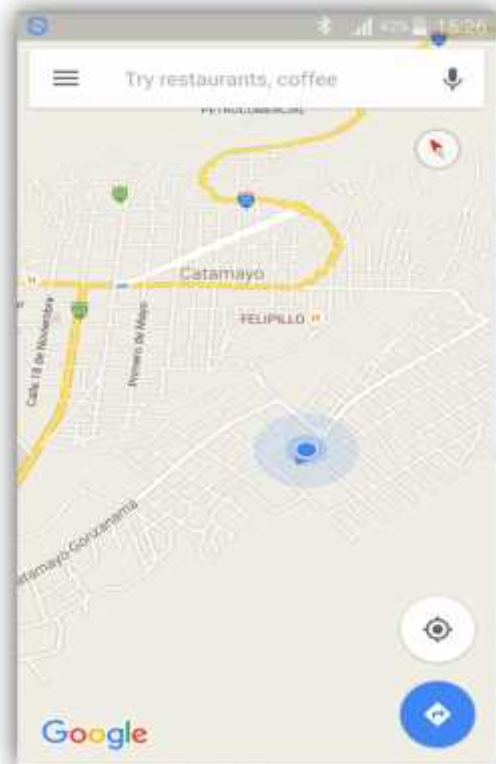


Figura 82. Prueba de validación 2 - Recorrido inter-cantonal: Punto de inicio.
Fuente: el autor.

De igual manera como en la prueba de validación no. 1, se ejecuta la aplicación la cual se conecta al scanner OBD-II automáticamente, extrayendo los datos y almacenando en el archivo CSV ya que en esta prueba se realizó con el dispositivo sin conexión internet.

Finalizado el recorrido, el punto de llegada exacto es mostrado en la Figura 85, que es una captura de la aplicación Google Maps.



Figura 83. Prueba de validación 2 - Recorrido inter-cantonal: Punto de llegada.
Fuente: el autor.

Para comprobar los datos, accedemos al almacenamiento interno del dispositivo, buscamos la carpeta llamada "OBD2" y localizamos el archivo "datos.csv".

A	B	C	D	E	F	G	
FECHA	TIEMPO	IDENTIFICADOR	LATITUD	LONGITUD	DESCRIPCION	COMANDO	DATA
2018-02-27	15:23:57	Xgl1234	-3.990371	-79.360079	Carga del motor	0104	410400
2018-02-27	15:23:57	Xgl1234	-3.990371	-79.360079	Temperatura del refrigerante	0105	410500
2018-02-27	15:23:57	Xgl1234	-3.990371	-79.360079	RPM	010C	410C0000
2018-02-27	15:23:57	Xgl1234	-3.990371	-79.360079	Velocidad	010D	410D0000
2018-02-27	15:23:57	Xgl1234	-3.990371	-79.360079	Presión del combustible	010A	NO DATA
2018-02-27	15:23:57	Xgl1234	-3.990371	-79.360079	Masa de flujo de aire MAF	0108	41080000
2018-02-27	15:23:57	Xgl1234	-3.990371	-79.360079	Temperatura ambiente	0146	NO DATA
2018-02-27	15:23:57	Xgl1234	-3.990371	-79.360079	Tipo de combustible	0151	415100
2018-02-27	15:23:57	Xgl1234	-3.990371	-79.360079	Tasa de combustible del motor	015C	NO DATA
2018-02-27	15:26:18	Xgl1234	-3.990371	-79.360079	Carga del motor	0104	410400
2018-02-27	15:26:18	Xgl1234	-3.990371	-79.360079	Temperatura del refrigerante	0105	410579
2018-02-27	15:26:18	Xgl1234	-3.990371	-79.360079	RPM	010C	410C0003
2018-02-27	15:26:18	Xgl1234	-3.990371	-79.360079	Velocidad	010D	410D0000
2018-02-27	15:26:18	Xgl1234	-3.990371	-79.360079	Presión del combustible	010A	NO DATA
2018-02-27	15:26:18	Xgl1234	-3.990371	-79.360079	Masa de flujo de aire MAF	0108	41080000
2018-02-27	15:26:18	Xgl1234	-3.990371	-79.360079	Temperatura ambiente	0146	NO DATA
2018-02-27	15:26:18	Xgl1234	-3.990371	-79.360079	Tipo de combustible	0151	415100
2018-02-27	15:26:18	Xgl1234	-3.990371	-79.360079	Tasa de combustible del motor	015C	NO DATA
2018-02-27	15:29:05	Xgl1234	-3.990371	-79.360079	Carga del motor	0104	410400
2018-02-27	15:29:05	Xgl1234	-3.990371	-79.360079	Temperatura del refrigerante	0105	410579
2018-02-27	15:29:05	Xgl1234	-3.990371	-79.360079	RPM	010C	410C23CD
2018-02-27	15:29:05	Xgl1234	-3.990371	-79.360079	Velocidad	010D	410D01

Figura 84. Prueba de validación 2 - Recorrido inter-cantonal: Datos almacenados en CSV.
Fuente: el autor.

La Figura 86 muestra parte de los datos almacenados en el archivo CSV durante la prueba de validación. El siguiente paso es subir los datos a la aplicación web para su visualización y descarga.

4.2.2.2. Aplicación Web

Una vez realizada la segunda prueba, procedemos a subir el archivo generado por la aplicación móvil. Ingresamos a la aplicación web, localizamos el área de subida de archivos y se sube el archivo CSV.



Figura 85. Prueba de validación 2 - Recorrido inter-cantonal: Importación del archivo CSV.
Fuente: el autor.

La Figura 88 muestra el mensaje presentado por la aplicación web una vez cargado el archivo CSV.

Para comprobar, se procede a realizar la respectiva consulta de datos:

- **Identificador:** lcg1234
- **Fechas desde y hasta:** 27-02-2016
- **Horas:** 15h00 – 17h00

La Figura 88 muestra el resultado devuelto por la aplicación web al realizar la consulta con los datos mencionados.



Figura 86. Prueba de validación 2 - Recorrido inter-cantonal: Resultados en la aplicación web.

Fuente: el autor.

Así mismo, debajo de esta interfaz se encuentran las opciones de descarga de los datos en formato CSV y JSON.

4.2.3. Prueba 3: Condiciones especiales

Para el desarrollo de esta prueba, se hizo un recorrido especial en dentro del área de prueba, la cual consistió en la obtención de los datos con el vehículo circulando dentro de un túnel (Túnel de los Ahorcados, Loja - Ecuador), se comprobó que bajo estas condiciones es difícil obtener la ubicación por geo-localización, debido a que depende mucho de los servicios de ubicación y el hardware del teléfono, ya que por lo general, estos dispositivos no están pensados para dar una alta precisión por temas de rendimiento de batería y los componentes básicos usados. Así mismo, existe la posibilidad de que exista un retardo de tiempo entre ubicaciones o algunas de estas no se tomen, lo que ocasionará que no se vean reflejadas en el mapa.



Figura 87. Prueba bajo condiciones especiales.
Fuente: el autor.

1.1. Pruebas de Aceptación

Las pruebas de aceptación fueron generadas a partir de los entregables elaborados durante los procesos de iteración y las dos pruebas de validación realizadas tanto para el aplicativo web como el aplicativo móvil.

En este apartado se analizan los entregables (módulos) generados en el desarrollo, tanto del aplicativo móvil como de la aplicación móvil.

1.1.1. Aplicación Móvil

Se realizan las pruebas de aceptación en base a los escenarios posibles que el usuario puede encontrar al momento de iniciar la interacción con la aplicación móvil.

Tabla 6. Prueba de Aceptación 1 – Conexión Bluetooth y GPS.

Caso de Prueba de Aceptación 01	
Código: PA01	Nombre: Conexión Bluetooth y GPS
Descripción: Cuando se inicia la aplicación, se debe comprobar el estado de los adaptadores Bluetooth y GPS.	
Condiciones de ejecución: El dispositivo dispone del hardware necesario para la ejecución.	
Entrada / Pasos de ejecución: <ul style="list-style-type: none"> • Inicia la aplicación. • Comprueba la existencia de los adaptadores. • Si existen, comprueba que estén activados. Opción 1: <ul style="list-style-type: none"> ○ Si están activados, procede a ejecutar los comandos. Opción 2: <ul style="list-style-type: none"> ○ Si están desactivados, muestra un mensaje para activar los adaptadores. 	
Resultado esperado: Los adaptadores Bluetooth y GPS están activados.	
Evaluación: Satisfactoria.	

Nota Fuente: el autor.

Tabla 7. Prueba de Aceptación 2 – Geo-localización y transmisión de datos OBDII.

Caso de Prueba de Aceptación 02	
Código: PA02	Nombre: Geo-localización y transmisión de datos OBDII
Descripción: Iniciada la aplicación, se ejecutan los comandos predeterminados.	
Condiciones de ejecución: El dispositivo dispone de los adaptadores Bluetooth, GPS activados.	
Entrada / Pasos de ejecución: <ul style="list-style-type: none"> • Se obtiene la ubicación. • La aplicación se conecta al scanner OBDII. • Se envían los comandos. • Se recibe la respuesta. • Se comprueba la conectividad internet. Opción 1: <ul style="list-style-type: none"> ○ Si hay conexión, procede a enviar los datos a la aplicación web para el almacenamiento. Opción 2: <ul style="list-style-type: none"> ○ Si no hay conexión, busca si existe el directorio de almacenamiento y el archivo CSV. ○ Si no existe, crea el directorio y el archivo. ○ Se edita el archivo CSV dentro del almacenamiento interno del dispositivo, generando una nueva fila de datos. <ul style="list-style-type: none"> • Se muestra el mensaje de confirmación de ejecución correcta de los comandos OBDII. 	
Resultado esperado: Opción 1:	

Se envían los datos a la aplicación web para su almacenamiento. Opción 2: Se almacenan los datos en el archivo CSV.
Evaluación: Satisfactoria.
Nota Fuente: el autor.

Tabla 8. Prueba de Aceptación 3 – Ejecución de la aplicación en segundo plano.

Caso de Prueba de Aceptación 03	
Código: PA03	Nombre: Ejecución de la aplicación en segundo plano.
Descripción: La aplicación continúa extrayendo datos incluso si se ha salido de ella.	
Condiciones de ejecución: <ul style="list-style-type: none"> ○ Se ha iniciado la aplicación. ○ No haber eliminado la aplicación de la multitarea del dispositivo. 	
Entrada / Pasos de ejecución: <ul style="list-style-type: none"> ● La aplicación está en ejecución. ● Se sale de la interfaz de usuario de la aplicación. ● La aplicación sigue funcionando mientras está en la multitarea del dispositivo. 	
Resultado esperado: Transmisión o almacenamiento de datos sin tener abierta la interfaz de usuario de la aplicación.	
Evaluación: Satisfactoria.	

Nota Fuente: el autor.

1.1.2. Aplicación Web

Se realizan las pruebas de aceptación en base a los escenarios posibles que el usuario puede encontrar al momento de iniciar la interacción con la aplicación web.

Tabla 9. Prueba de Aceptación 4 – Consulta de recorridos.

Caso de Prueba de Aceptación 04	
Código: PA04	Nombre: Consulta de recorridos
Descripción: Se presenta el formulario de consulta en donde el usuario puede consultar los recorridos en base a un rango de fecha y tiempo.	
Condiciones de ejecución: Ingreso a la aplicación web.	
Entrada / Pasos de ejecución: <ul style="list-style-type: none"> ● Ingresa a la aplicación web. ● Identifica la sección de consultas. ● Ingresa el campo identificador. ● Selecciona una fecha de inicio de la consulta. ● Selecciona una fecha de fin de la consulta. ● Selecciona una hora de inicio de la consulta. ● Selecciona una hora de fin de la consulta. ● Presiona el botón “Enviar”. 	

<p>Resultado esperado:</p> <ul style="list-style-type: none"> • Se presenta el mapa con los recorridos realizados: marcadores interconectados marcando una ruta. • Cada marcador posee como información la fecha y hora en que fue registrado. • Se presenta las opciones de descarga de la información consultada.
<p>Evaluación: Satisfactoria.</p> <p>Nota Fuente: el autor.</p>

Tabla 10. Prueba de Aceptación 5 – Subida de datos.

Caso de Prueba de Aceptación 05	
Código: PA05	Nombre: Subida de datos
Descripción: Se presenta el formulario de envío de un archivo con extensión CSV.	
Condiciones de ejecución: Ingreso a la aplicación web.	
Entrada / Pasos de ejecución: <ul style="list-style-type: none"> • Ingresa a la aplicación web. • Identifica la sección de carga de datos. • Ingresa el campo identificador. • Selecciona un archivo de formato CSV generado por la aplicación. • Presiona el botón “Subir Datos”. 	
Resultado esperado: <p>Opción 1:</p> <ul style="list-style-type: none"> • Se presenta una página con el mensaje de confirmación del envío e importación correcta de datos. Se muestra un enlace hacia la página principal. <p>Opción 2:</p> <ul style="list-style-type: none"> • Se presenta una página con el mensaje de error. Se muestra un enlace hacia la página principal. 	
Evaluación: Satisfactoria.	
Nota Fuente: el autor.	

Tabla 11. Prueba de Aceptación 6 – Descarga de datos.

Caso de Prueba de Aceptación 06	
Código: PA06	Nombre: Descarga de datos
Descripción: Se presenta la opción de descarga de los datos consultados en formato CSV, JSON.	
Condiciones de ejecución: Ingreso a la aplicación web. Haber realizado una consulta de un recorrido.	
Entrada / Pasos de ejecución: <ul style="list-style-type: none"> • Se muestra la sección de descarga. • Se muestra el botón “CSV”. • Se muestra el botón “JSON”. • Da clic en cualquier botón. 	
Resultado esperado:	

<p>Opción 1:</p> <ul style="list-style-type: none">• Se descarga un archivo de extensión CSV con los datos consultados. <p>Opción 2:</p> <ul style="list-style-type: none">• Se descarga un archivo de extensión JSON con los datos consultados.
<p>Evaluación: Satisfactoria.</p>

Nota Fuente: el autor.

1.2. Pruebas de Rendimiento

Las pruebas de rendimiento se realizaron para determinar el rendimiento de las aplicaciones, tanto web como móvil.

1.2.1. Aplicación Móvil

Dentro de la aplicación móvil se evaluaron los siguientes aspectos:

- Consumo de datos móviles.
- Espacio de almacenamiento interno.
- Uso de memoria RAM.

Para determinar estos aspectos, se hizo uso de las herramientas nativas del sistema operativo Android.

1.2.1.1. Consumo de datos móviles

Según el estimado proporcionado por la herramienta de uso de datos, la aplicación móvil tiene un consumo mínimo de datos, la Figura 89 muestra una captura de la gestión de datos efectuada en una prueba aislada, en la cual se hizo el mismo recorrido de la prueba de validación no. 2 cuya ruta comprende un estimado de 38 km.

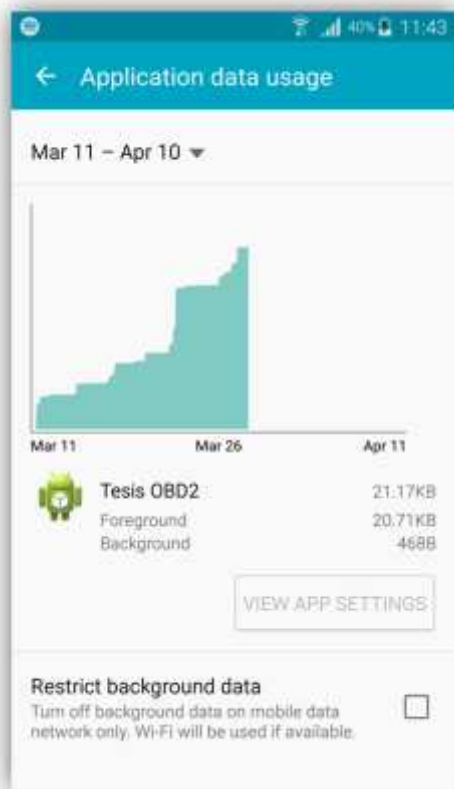


Figura 88. Pruebas de rendimiento - Aplicación móvil: Consumo de datos móviles.
Fuente: el autor.

El consumo de datos realizado es de 21.17 KB en un recorrido de 40 minutos.

1.2.1.2. Espacio de almacenamiento interno

Según la herramienta de gestión de aplicaciones de Android, la aplicación tiene un tamaño de 15.08 MB en total, una vez instalada. La Figura 90 muestra a detalle el espacio usado por la aplicación.



Figura 89. Pruebas de rendimiento - Aplicación móvil: Tamaño de la aplicación.
Fuente: el autor.

1.2.1.3. Uso de memoria RAM

El uso continuo de la aplicación representa el consumo de memoria RAM del dispositivo móvil, por lo que el dispositivo debería contar con una cierta cantidad de memoria RAM libre para la ejecución de aplicaciones.

Normalmente, el sistema operativo Android consume entre el 50-70 % de memoria RAM de un dispositivo, lo restante está destinado para el uso de aplicaciones.

La Figura 91 muestra el consumo de memoria RAM de la aplicación, cuyo valor es aproximadamente 24 MB, un consumo realmente bajo.

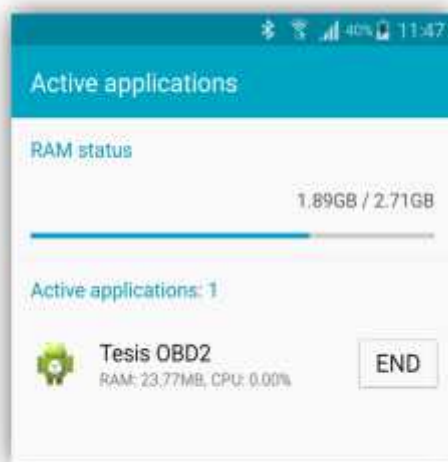


Figura 90. Pruebas de rendimiento - Aplicación móvil: Consumo de memoria RAM.

Fuente: el autor.

1.2.2. Aplicación Web

Dentro de la aplicación web se evaluaron los siguientes aspectos:

- Velocidad de carga.
- Validación de código HTML.
- Validación de estilos CSS.

Para determinar estos aspectos, se hizo uso de herramientas en línea.

1.2.2.1. Velocidad de carga

Para el cálculo de la velocidad, se hizo uso de la herramienta PINGDOM(Pingdom 2015). La Figura 92 muestra el resultado de la prueba.



Figura 91. Pruebas de rendimiento - Aplicación web: Velocidad de carga.
Fuente: el autor.

El resultado muestra que:

- La carga del sitio demora 465 milisegundos.
- Se hace referencia a 8 recursos (Script, CSS).
- El peso de la página inicial es de 281.4 KB.

1.2.2.2. Validación de etiquetas HTML

Para evaluar la estructura HTML se usó el validador de la W3C(W3C 2016a), el cual permite detectar anomalías en el etiquetado HTML.



Figura 92. Pruebas de rendimiento - Aplicación web: Validación de código HTML.
Fuente: el autor.

El resultado muestra que la estructura HTML de la aplicación web es correcta.

1.2.2.3. Validación de estilos CSS

Para validar la estructura de estilos CSS también se usa la herramienta de la W3C(W3C 2016b).



Figura 93. Pruebas de rendimiento - Aplicación web: Validación de estilos CSS.
Fuente: el autor.

La Figura 94 muestra el resultado de validar los estilos CSS. Existen 38 errores, sin embargo hacen referencia a las librerías usadas como es jQuery, Bootstrap. Estos errores no afectan directamente el rendimiento de la aplicación web.

CONCLUSIONES

Finalizado el proyecto, se puede llegar a las siguientes conclusiones:

- Se ha logrado extraer los datos de los sensores del vehículo en tiempo real creando una aplicación móvil, compatible con dispositivos Android versión 4.0 o superior, equipados con GPS e interfaz Bluetooth.
- Debido a que un scanner OBD-II es un dispositivo genérico, no se garantiza que la aplicación móvil funcione en todos los vehículos que dispongan la entrada de diagnóstico OBD-II, la causa es el hardware interno, ya que varía en las versiones de chips usados para su construcción.
- Los datos extraídos de carga del motor, temperatura del refrigerante, rpm, velocidad, presión del combustible, flujo de combustible, masa de flujo de aire, y temperatura ambiente, permiten estimar el consumo de combustible y obtener un diagnóstico general del vehículo.
- Se implementó una aplicación web usando tecnologías PHP, JS, Bootstrap, API Google Maps, consiguiendo visualizar las ubicaciones obtenidas mediante geolocalización y descargar los datos obtenidos desde el dispositivo móvil.
- Se ha determinado que los datos de ubicación y su transmisión al servidor dependen de factores como: hardware del dispositivo, interferencias físicas como túneles o edificios, disponibilidad de los proveedores de servicios de ubicación y la cobertura de los operadores móviles.
- El uso de Heroku como servicio basado en la nube permite el despliegue de aplicaciones, tanto para hosting de la aplicación web como para motor de base de datos, con su complemento ClearDB.

RECOMENDACIONES

En base al desarrollo de este proyecto y a las conclusiones, se plantean algunas recomendaciones en caso de futuras implementaciones y/o trabajos relacionados:

- Hacer uso de la documentación oficial y usar la versión más reciente del SDK de Android, para mayor compatibilidad con dispositivos.
- Elegir un scanner OBD-II el cual posea una sola interfaz (Bluetooth) para trabajar en la plataforma de Android.
- Evaluar las diferentes opciones de scanner OBDII, ya que no existe una sola marca o fabricante.
- Investigar la versión de componentes del scanner OBDII tipo Bluetooth, ya que esto determinará la compatibilidad con los vehículos, mientras más nueva sea la versión, podrá funcionar correctamente con vehículos modernos. Los distribuidores de estos productos informan con qué año y modelos de vehículos son compatibles.
- Desarrollar aplicaciones Android bajo una metodología ágil, por la volatilidad de los requerimientos y ajuste de cambios.
- Desplegar aplicaciones web en servicios en la nube como Heroku, ya que ofrecen opciones especializadas de escalabilidad y disponibilidad del 99% del tiempo, ajustándose tanto a pequeños como grandes proyectos y extendiendo la funcionalidad mediante complementos.

TRABAJOS FUTUROS

El trabajo realizado puede ser usado como base para futuras investigaciones y aplicaciones, algunas ideas se exponen a continuación:

- Aplicaciones de seguridad: monitoreo en tiempo real de la ubicación de un vehículo agregando consultas recursivas sobre la aplicación web y extender las funciones para diferenciar cada recorrido en el mapa.
- Minería de datos: procesar los datos para determinar estilos de conducción.
- Aplicaciones móviles, extendiendo las funciones de la aplicación Android para obtener parámetros más avanzados del estado del vehículo.
- Aplicaciones de generación de alertas, para optimización del consumo de combustible y recursos del vehículo.

BIBLIOGRAFÍA

- Abrigo Maldonado, Alexander. 2007. "Compendio Del Sistema OBDII." <http://dspace.ups.edu.ec/bitstream/123456789/1142/21/Tesis.pdf>.
- Aguirre Chacón, Luis, and Huber Sinche Ricra. 2013. "Diseño de Una Aplicación Móvil Para La Consulta Académica de La FIIS-UTP." http://pis1.wikispaces.com/file/view/Presentacion+Final_Tesis+I.pdf.
- Android Developers. 2011. "What Is Android."
- . 2016a. "ADT Plugin Release Notes - Android Studio." <https://developer.android.com/studio/tools/sdk/eclipse-adt.html>.
- . 2016b. "Develop Apps - Android Developers." <https://developer.android.com/develop/index.html>.
- Brahler, S. 2010. *Analysis of the Android Architecture*.
- Cáceres, Carlos, and Marlon García. 2012. "Implementación de Un Banco de Pruebas de Inyección Electrónica de Un Motor Corsa 1.4lt OBD II Para El Taller de La Escuela de Ingeniería Automotriz de La Escuela Superior Politécnica de Chimborazo." <http://dspace.epoch.edu.ec/bitstream/123456789/2313/1/65T00064.pdf>.
- Cobo, Angel, Patricia Gómez, Daniel Pérez, and Rocío Rocha. 2005. *PHP Y MySQL: Tecnología Para El Desarrollo de Aplicaciones Web*. Spain: Diaz de Santos.
- Eclipse Foundation. 2016. "Mars Eclipse." <https://eclipse.org/mars/>.
- EPA, US. 2015a. "Basic Information - On-Board Diagnostics (OBD)." <http://www.epa.gov/obd/basic.htm>.
- . 2015b. "Basic Information | On-Board Diagnostics (OBD) | US EPA." <https://www3.epa.gov/obd/basic.htm>.
- Equipo Vértice. 2009. *Diseño Básico de Páginas Web En HTML*. Málaga: Publicaciones Vértice.
- Google. 2016a. "BluetoothDevice - Android Developers." <http://developer.android.com/intl/es/reference/android/bluetooth/BluetoothDevice.html>.
- . 2016b. "Location Strategies - Android Developers." <http://developer.android.com/intl/es/guide/topics/location/strategies.html>.
- Google Developers. 2016a. "Google Maps JavaScript API - Google Developers." <https://developers.google.com/maps/documentation/javascript/?hl=es>.
- . 2016b. "Marcadores - Google Maps JavaScript API - Google Developers." <https://developers.google.com/maps/documentation/javascript/markers>.
- Heroku. 2016a. "ClearDB MySQL - Heroku Dev Center." <https://devcenter.heroku.com/articles/cleardb>.
- . 2016b. "Cloud Application Platform - Heroku." <https://www.heroku.com/>.
- . 2016c. "Heroku Toolbelt." <https://toolbelt.heroku.com/>.
- Juric, Radmila. 2000. "Extreme Programming and Its Development Practices." *22nd Int conference on information technology interfaces (0)*.
- Kappel, G, B Pröll, S Reich, and W Retschitzegger. 2006. *Zhurnal Eksperimental'noi i Teoreticheskoi Fiziki Web Engineering: The Discipline of Systematic Development of Web Applications*. <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:No+Title#0\nhttp://wwwcs.uni-paderborn.de/cs/ag->

engels/ag_dt/Courses/Lehrveranstaltungen/WS0607/WE/slides/WebEngineering-WT-2006-Chapter-01-Introduction(2).pdf.

- Van Lancker, Luc. 2009. *XHTML Y CSS: Los Nuevos Estándares Del Código Fuente*. 2da. ed. Barcelona: Ediciones ENI.
- Luján Mora, Sergio. 2002. *Programación de Aplicaciones Web: Historia, Principios Básicos Y Clientes Web*. ed. Editorial Club Universitario. Alicante: Gamma.
- Meseguer, Javier. 2012. "Caracterización de Los Estilos de Conducción Mediante Smartphones, Dispositivos Obd-li Y Redes Neuronales." https://riunet.upv.es/bitstream/handle/10251/21025/JavierMeseguerAnastasioTesisnaMaster_MIC.pdf?sequence=1&isAllowed=y.
- Pingdom. 2015. "Website Speed Test." <http://tools.pingdom.com/fpt/>.
- Reitze, Arnold W. 2001. *Air Pollution Control Law: Compliance and Enforcement*. Environmental Law Institute. <https://books.google.com/books?id=M8w5yJbNTD0C&pgis=1> (April 27, 2016).
- Ruhe, M., R. Jeffery, and I. Wiczorek. 2003. "Using Web Objects for Estimating Software Development Effort for Web Applications." In *Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No.03EX717)*, IEEE Comput. Soc, 30–37. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1232453>.
- Shi, Zhong et al. 2011. "Agile Planning and Development Methods." *Computer Research and Development (ICCRD), 2011 3rd International Conference on* 1: 488–91. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5764064> <http://ieeexplore.ieee.org/pdf/1989159266/Shi-2011-Agile-planning-and-d.pdf> <http://ieeexplore.ieee.org/jelx5/5756602/5763958/05764064.pdf?tp=&arnumber=5764064&isnumber=5763958>.
- W3C. 2016a. "The W3C Markup Validation Service." <https://validator.w3.org/>.
- . 2016b. "Validación de CSS Del W3C." <https://jigsaw.w3.org/css-validator/>.
- Zaldivar, Jorge, Carlos T. Calafate, Juan Carlos Cano, and Pietro Manzoni. 2011. "Providing Accident Detection in Vehicular Networks through OBD-II Devices and Android-Based Smartphones." *Proceedings - Conference on Local Computer Networks, LCN*: 813–19.

ANEXOS

Anexo 1. Consumo del API Google Maps.

Para realizar el consumo de las APIs de Google, es necesario tener una cuenta de Google, que permite acceder a un sinnúmero de servicios como correo, ofimática en línea, almacenamiento en la nube, etc.

Para obtener acceso a las APIs, una vez autenticado en Google, se accede a la siguiente dirección:

<https://console.developers.google.com>

Se muestra la página con los accesos a las diferentes APIs de los servicios de Google, aquí en la categoría APIs de Google Maps se usa el API Google Maps JavaScript.

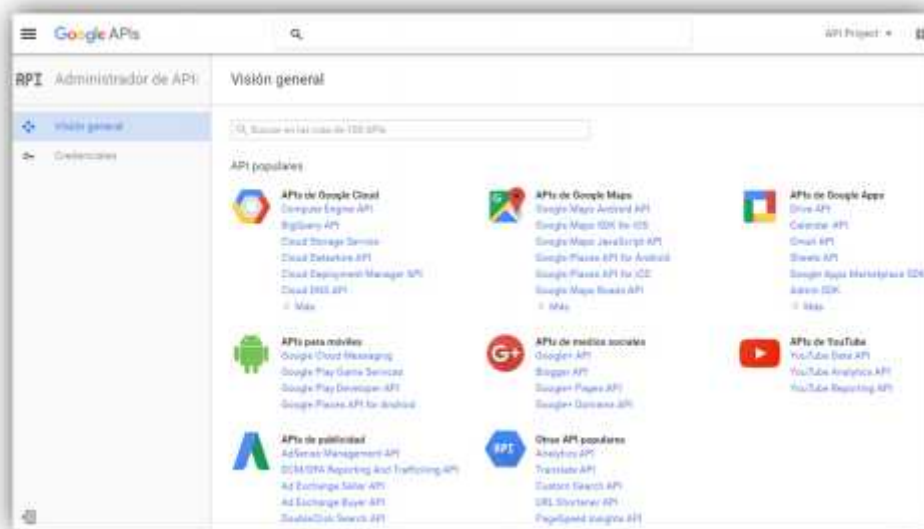


Figura 94. Consola de Desarrollo de Google.
Fuente: el autor.

Al seleccionar el API, Google provee información del uso y el porcentaje admitido para el mes ya que el servicio gratuito posee un límite de uso por día y mes.



Figura 95. Descripción del API JavaScript Google Maps.
Fuente: el autor.

Para activar este servicio, se da clic en el botón Habilitar de la parte superior, con lo cual el API queda activo y listo para procesar las peticiones.

Las peticiones que acepta son mediante una clave que debe ser generada como una credencial, para ello en la parte izquierda existe el menú Credenciales en donde se gestionan los accesos.

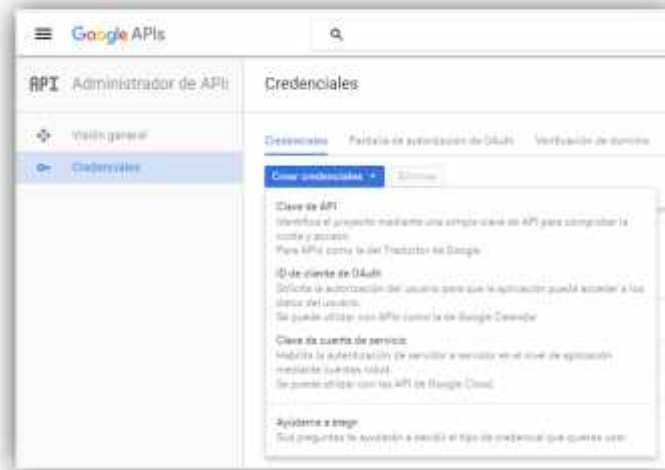


Figura 96. Obtención de la clave de acceso al API Google Maps.
Fuente: el autor.

Al seleccionar la opción de “Crear Credenciales”, se muestran tres formas de acceso a los servicios, dependiendo de cómo se llame al API desde la aplicación, se debe optar por alguna de ellas, en este caso se usará la opción “Clave de API” que permite tener una clave cifrada la cual se incluye en la llamada al API.



Figura 97. Creando la clave de acceso al API Google Maps.
Fuente: el autor.

La Figura 96 muestra las opciones de creación de la clave, en este caso se procede a crear una clave de Navegador, que será quien gestione las peticiones desde la aplicación directamente hacia los servidores de Google.

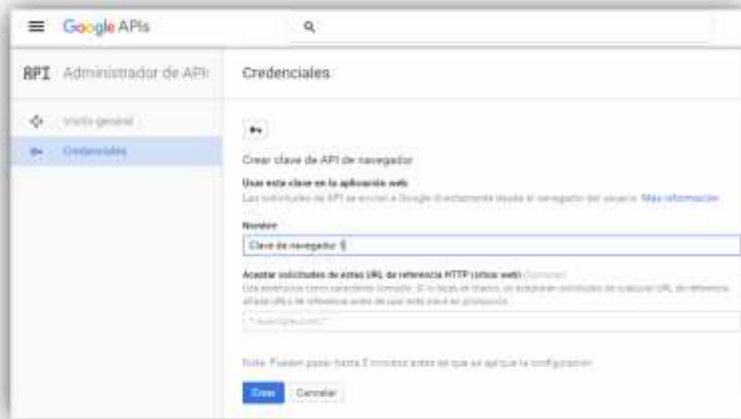


Figura 98. Clave de Navegador API Google Maps.
Fuente: el autor.

Se procede a introducir un nombre que identifique a la aplicación y de forma opcional se puede filtrar las solicitudes agregando los sitios desde los cuales se autorice hacer la llamada al API, se deja en blanco por defecto para aceptar solicitudes desde cualquier sitio.

Al pulsar el botón crear, se genera la clave la cual está lista para ser incluida en las llamadas al API.

Anexo 2. Hosting de aplicaciones en la plataforma de Heroku.

Heroku es una plataforma de despliegue de aplicaciones, con herramientas en la nube que permiten de forma fácil y gratuita (hasta cierto punto) el poder implementar aplicaciones y servicios en distintas tecnologías que van desde lenguajes como PHP, Python, Ruby, etc. Hasta servicios de mail y bases de datos.

Para hacer uso de estos servicios es necesario crear una cuenta en Heroku y descargar la herramienta Heroku Toolbelt(Heroku 2016c)que permite mediante línea de comandos gestionar aplicaciones para desplegar en Heroku, así mismo como un repositorio GIT para control de versiones.

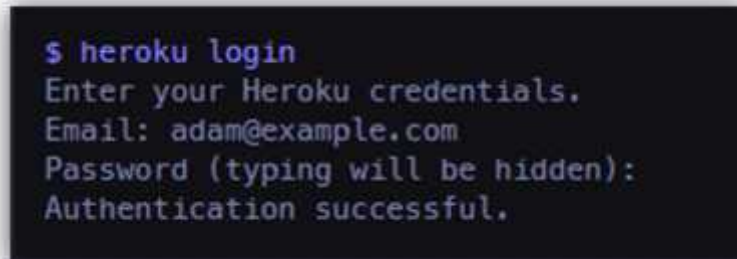
Hosting en Heroku

Para usar el hosting de Heroku, primero se debe crear en local un directorio en donde se coloquen los archivos de la aplicación, adicionalmente se crea un repositorio mediante GIT en donde se registren los cambios que ocurran en cada uno de los archivos.

Se inicia el cliente Heroku Toolbelt el cual es una ventana de ejecución de comandos, primero se tiene que iniciar sesión en la cuenta de Heroku, mediante el comando:

```
heroku login
```

Al ingresar el comando, se pedirá el email y password de la cuenta, luego de ingresar estos datos se mostrará el mensaje de autenticación exitosa.



```
$ heroku login
Enter your Heroku credentials.
Email: adam@example.com
Password (typing will be hidden):
Authentication successful.
```

Figura 99. Autenticación en Heroku Toolbelt.
Fuente: recuperado de <https://goo.gl/g8uXAE>

El siguiente paso es crear una aplicación, para ello se ejecuta el comando:

```
heroku create
```

Adicionalmente, se puede agregar un nombre, el cual determinará el subdominio al cual se podrá acceder a la aplicación, por ejemplo:

```
heroku create tesisobd
```

Con ello, se mostrará el mensaje con la URL para acceder a la aplicación, dependiendo del caso si no se especificó un nombre, tomará un nombre aleatorio bajo el subdominio herokuapp.com como se muestra en la Figura 99.

```
$ cd ~/myapp
$ heroku create
Creating stark-fog-398... done, stack is cedar-14
http://stark-fog-398.herokuapp.com/ | https://git.heroku.com/stark-fog-398.git
Git remote heroku added
```

Figura 100. Creación de una aplicación en Heroku.
Fuente: recuperado de <https://goo.gl/g8uXAE>

Ahora, procedemos a crear el repositorio GIT y enviar todos los archivos de la aplicación a Heroku, para ello se usan los comandos:

git init

git add .

git commit -m "primer envío"

git push heroku master

heroku open

El primer comando genera el repositorio GIT, el segundo comando agrega todos los archivos (o los que contengan cambios) para su envío, el tercer comando permite agregar una descripción del envío que se realiza, el cuarto comando envía los archivos (o los cambios realizados) al hosting de heroku, el quinto comando abre el navegador web con la dirección de la aplicación para su visualización.

La plataforma de Heroku posee tal dinamismo que reconoce el tipo de aplicación y compila los recursos necesarios para poder ejecutarla. Una vez hecho esto, si se quiere realizar cambios en el código, basta con enviarlos nuevamente usando los comandos anteriormente mencionados¹⁵.

Servidor de Base de Datos: ClearDB

Heroku posee extensiones que pueden ser añadidas a las aplicaciones que se despliegan en su plataforma. Entre estos servicios está ClearDB, un servidor en la nube de base de datos MySQL.

Para hacer uso de esta extensión, hay que usar el cliente Heroku Toolbelt, autenticarse en la cuenta¹⁶ y ejecutar el siguiente comando:

heroku addons:create cleardb:ignite

Con esto, el servicio estará agregado a la aplicación, seguidamente se extrae la información de la conexión generada, la cual contiene el usuario y la clave para acceder a la base de datos:

¹⁵ Nota: Se debe omitir el comando de inicialización git init.

¹⁶ Ver Anexo 2. Hosting en Heroku.

```
heroku config | grep CLEARDB_DATABASE_URL
```

El comando anterior genera una línea de este estilo:

```
mysql://usuario:clave@host/heroku_db?reconnect=true
```

Donde se deben extraer los datos de usuario, clave y host para agregar en la conexión de la aplicación que se esté codificando, o si se quiere usar un cliente de gestión de base de datos remoto.

Anexo 3. Hardware y circuitos OBD-II.

A día de hoy existen variedad de fabricantes de hardware OBD-II, muchos de estos dispositivos son scanner con varias interfaces, las cuales pueden ser WiFi, Bluetooth, USB, llegando a tener todas estas en un solo scanner.

La mayoría de distribuidores en los sitios web de comercio electrónico sugieren que los scanner OBD-II deben ser usados en vehículos desde el año 2003 en adelante, esto se puede comprobar realizando la búsqueda del scanner en tiendas como Amazon, Ebay, Mercado Libre, Aliexpress, entre otras.

Scanner OBD-II ELM327.

La mayoría de scanner disponibles usan en su construcción el chip Bluetooth ELM327, este es usado para la conexión entre puertos OBD-II y RS232. El EML327 puede manejar todos los protocolos OBD incluidos los más nuevos protocolos en los vehículos más modernos. ELM327 usa ASCII para la comunicación con el puerto OBD-II.

ELM327 actúa como una interface de línea de comandos, en su respuesta incluirá el carácter '>' en el puerto serial, además no ejecutará un comando hasta que se detecte un salto de línea o break.

Comunicación con el vehículo.

El estándar requiere que un comando OBD-II sea acompañado del modo de trabajo, este modo describe el tipo de dato que se solicita. El primer byte es el modo, el segundo el parámetro de identificación conocido como PID. Los modos y comandos están bien referenciados en los documentos SAE J1979 o ISO 15031-5.

No se requiere que los vehículos soporten todos los modos de trabajo, y con los modos también los PID's (algunos de los primeros vehículos OBD-II solo soportaban un número limitado de ellos). Con cada modo, el PID 00 es reservado para mostrar que PID's soporta el vehículo en ese modo.

Para saber la versión de chip ELM327 que se está usando, basta con enviar el comando "atz", la cual arroja la versión que está usando, algo como "ELM327 v2.1". Mientras más nueva sea la versión, tendrá mayor compatibilidad con los últimos modelos de vehículos.

Para enviar los comandos, se procede de a enviar el modo de trabajo seguido del PID:

```
>01 00
```

Recibirá una respuesta en formato decimal:

```
>41 00 BE 1F B8 10
```

El primer número representa la respuesta del modo de trabajo (41), el segundo repite el PID que se ha consultado, los siguientes 4 bytes representan la respuesta enviada

por el sensor, en formato hexadecimal. Este valor debe ser convertido a su equivalente decimal para tener un formato legible.

Para tener información a detalle, se adjunta el documento técnico referencial en idioma inglés de Elmelectronics, con información de modos de trabajo, comandos y diagramas de los circuitos ELM327.



We are often asked how to connect our ELM327 to a Bluetooth wireless system. In addition to working at speeds of up to 700kbps, Bluetooth offers several other advantages over wired interfaces. For example, it provides galvanic isolation between the vehicle and the computer's wiring, and it also allows connections with devices that do not have RS232 or USB ports.

The following provides a few ideas on how you might add Bluetooth to your ELM327 circuit.

Using Serial to Bluetooth Adapters

There are a few RS232 to Bluetooth converters on the market that can simply connect to a 9 pin RS232 connector, and provide Bluetooth capability. Two of these devices are the Firefly from Roving Networks (www.rovingnetworks.com) and the GBS301 from IOGEAR (www.iogear.com).

Both the FireFly and the IOGEAR provide a connector for an external power source, but also allow DC to be provided through the RS232 connector (on pin 9). Your ELM327 circuit can be modified as shown in Figure 1 to provide this power, so that you do not need to connect the DC supply each time:

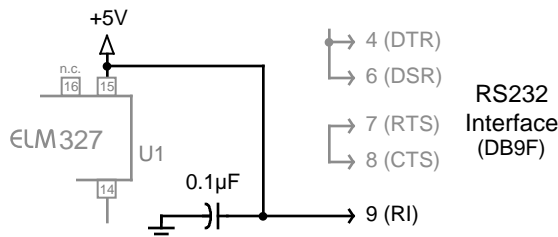


Figure 1. 5V for the Bluetooth Adapter

Note that these modules draw a significant current when operating (about 50mA), so you may need to modify your ELM327's 5V supply in order to provide the required power. If your circuit uses a small 78L05 regulator (as is shown in Figure 9 of the ELM327 data sheet), we recommend that you change it to a larger 7805 type regulator.

If you are using the RS232 interface from the ELM327 data sheet, you will find that it does not always work reliably at 115200 baud, so should only be used at speeds of 57600 or less. The IOGEAR GBS301 offers 9600, 19200, 38400, and 57600 baud rates from the dip switch, so can be easily set for 38400 or 57600 baud and connected to your ELM327 circuit. The FireFly adapter only offers 9600 or 115200 baud from the dip switch, however, which means that

you'll have to set it for 9600 baud (or learn to program it). Decide on which rate you prefer, and then make sure that you set the ELM327's pin 6 logic level to allow it.

Regardless of which Bluetooth adapter that you choose, the next step is to 'pair' the device with your computer. Follow the manufacturer's instructions on how to do this. Do not be surprised if there is no software needed, as many modern operating systems will have default Bluetooth drivers built in. Simply follow the procedure to add the device (as a 'Serial Port') and then use it as you would any physically connected serial device.

Using Serial to Bluetooth Modules

Connecting a Bluetooth adapter to the RS232 connector may be convenient, but your connection speeds are severely limited by the RS232 circuitry. If you could bypass the RS232 stage and connect the ELM327 directly to the Bluetooth transceiver, then you would be able to operate at much higher baud rates.

Several modules that contain a complete serial Bluetooth system (including an antenna) are now available in the marketplace. These provide a logic level RS232 interface for your circuitry and a standard Bluetooth protocol for 'connecting' to your controlling device. One such module is the LMX9838 from National Semiconductor (www.national.com).

The LMX9838 is a terrific little (1.0cm x 1.7cm) solution, but it needs a 3.3V supply to operate. Since the ELM327 uses 5V, it would seem that the two are not compatible. We often get questions concerning such an interface, so will discuss it here.

Interfacing to 3.3V

Connecting to a 3.3V system usually requires some form of voltage conversion. Some 3.3V ICs have inputs that are '5V tolerant' (but not all), and most 5V logic (the ELM327 included) will not tolerate 3.3V on an input, so in general a circuit is required.

You can buy 'level translator' integrated circuits to handle the voltage translation for you. Examples are the TXB0102 by Texas Instruments (www.ti.com), and the ST2129 by ST Microelectronics (www.st.com). Their connection is straight-forward as can be seen in the bubble of Figure 3.

If you wish to build your own interface circuit, it can be done with commonly available parts. Figure 2 shows one example that works well for most of the

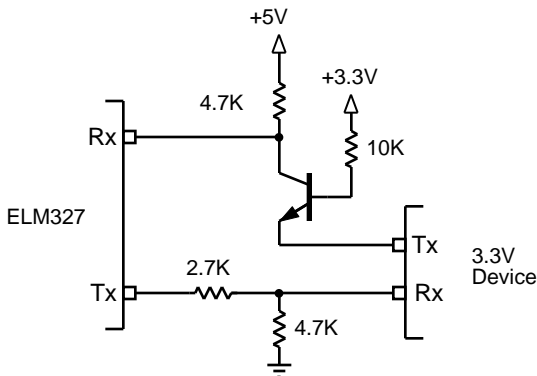


Figure 2. A 5V to 3.3V Interface

baud rates that you would encounter with an ELM327.

This circuit uses a simple resistor voltage divider to reduce the ELM327's 5V Tx output to approximately 3.3V. The 2.7K and 4.7k resistors shown work well over a wide range of frequencies while still presenting a reasonable load to the ELM327.

Interfacing a 3.3V output to the ELM327's 5V input requires a little more circuitry. Amplification is needed to increase the 3.3V level, so we add a transistor to the circuit, as shown. When the Tx output from the '3.3V Device' is above about 2.8V, the transistor will be turned off, and the ELM327's Rx input will have 5V applied to it through the 4.7k resistor. When the 3.3V output switches to a low level, the transistor will be driven into saturation and the Rx input will go to about 0V.

Since the transistor is operated in saturation, there will be a slight delay in turning off due to the charge storage. Because of this delay, the rising edge of the ELM327 Rx input will show a very slight pause in the voltage while the charge is depleted. This does not really affect the operation here, but if you wish to adapt this circuit for use at higher frequencies, it is something that you must be aware of.

Connecting to the Bluetooth module

The next page shows an LMX9838 Bluetooth module connected to an ELM327 through the circuit of Figure 2. (The ELM327 circuit is from Figure 9 of the ELM327 Data Sheet.)

The module requires a 3.3V power supply, so a voltage regulator has been added. We show an MIC5200-3.3 from Micrel (www.micrel.com), but there are many alternatives (such as Microchip's MCP1700). The regulator has been connected directly to the +5V supply, and filter capacitors have been connected to

its output. To meet the extra current requirements of the module (65mA maximum), be sure that your 5V supply uses a 7805 type regulator, and not a 78L05 type.

The LMX9838 module can be set for 9600 or 115200 baud by changing the logic levels on pins 25 and 26. Initially, set the module for 9600 baud, but make provisions for switching to 115200 baud later (as 9600 baud is too slow).

The ELM327 comes from the factory set for 9600 and 38400 baud - to provide a rate of 115200 requires changing the 38400 baud rate with Programmable Parameter 0C.

To change the ELM327's baud rate, first connect the circuit of Figure 3 for 9600 baud (see the chart for the connections). Power your circuit, and set up your PC to use Bluetooth for a serial port. Follow the Operating System's procedure to 'pair' to the LMX9838 (the code is 0000), and then connect to the ELM327 through the new serial port at 9600 baud. You should be able to perform simple commands such as AT I, even if no vehicle is connected:

```
>AT I
ELM327 v1.4b
```

Once you know that all is working well at 9600 baud, set the ELM327 for a power on default baud rate of 115200 (when pin 6 is high). To do this, assign the value 23 to PP 0C:

```
>AT PP 0C SV 23
```

and then enable the use of this new value:

```
>AT PP 0C ON
```

That's all that is required is to permanently change the baud rate. Now, change the wiring so that the ELM327 and the LMX9838 power up to the high speed baud rate. That is, connect pin 6 of the ELM327 to 5V, and connect the resistors on the LMX9838 module pins 25 and 26 to 0V and 3.3V, respectively.

Apply power to your circuit, and test the Bluetooth connection. You may have to change your software for 115200 baud instead of 9600. If all has gone well, you should now have a high speed wireless OBD interpreter.

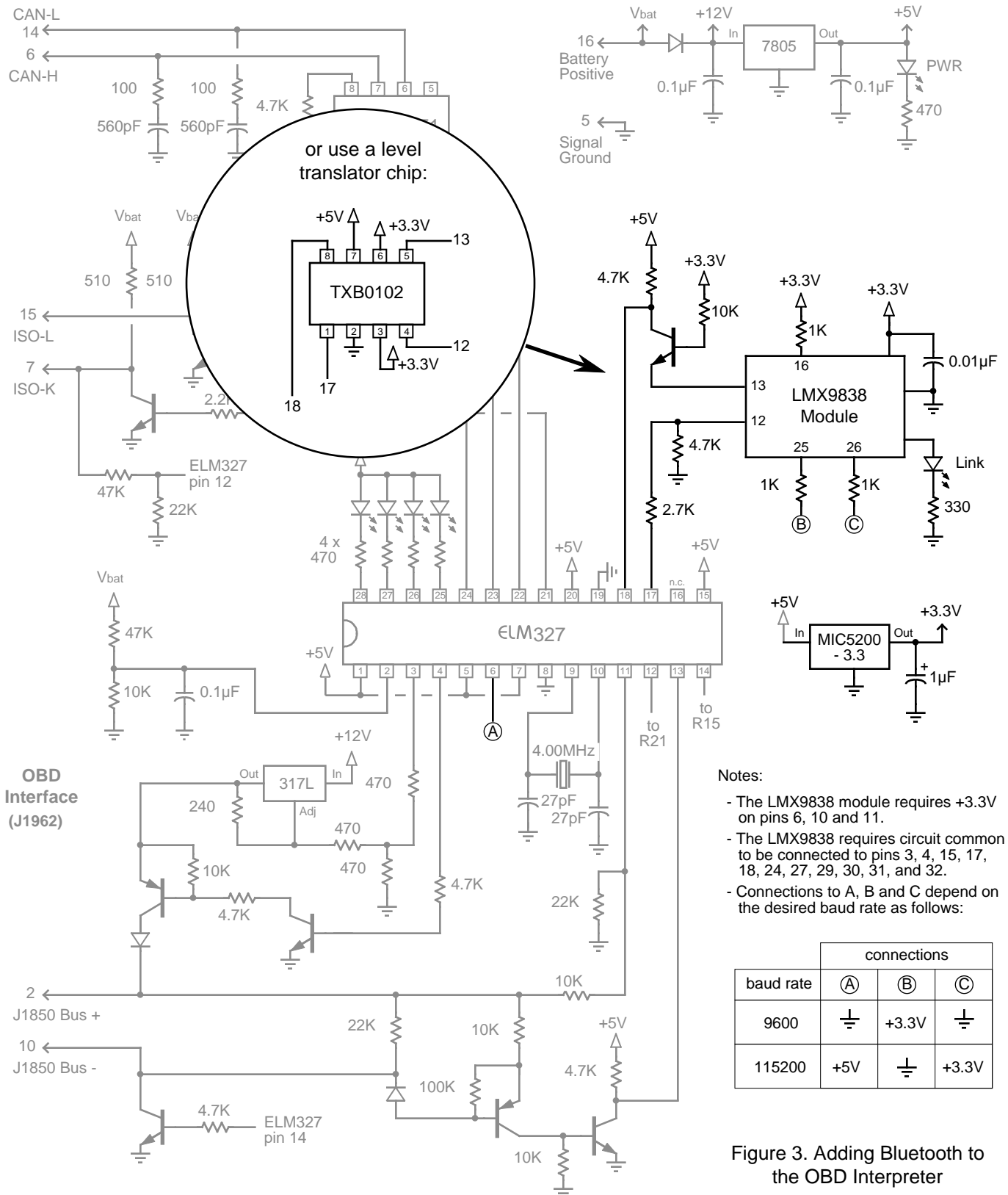


Figure 3. Adding Bluetooth to the OBD Interpreter



Description

Almost all of the automobiles produced today are required, by law, to provide an interface for the connection of diagnostic test equipment. The data transfer on these interfaces follow several standards, but none of them are directly usable by PCs or smart devices. The ELM327 is designed to act as a bridge between these On-Board Diagnostics (OBD) ports and a standard RS232 serial interface.

In addition to being able to automatically detect and interpret nine OBD protocols, the ELM327 also provides support for high speed communications, a low power sleep mode, and the J1939 truck and bus standard. It is also completely customizable, should you wish to alter it to more closely suit your needs.

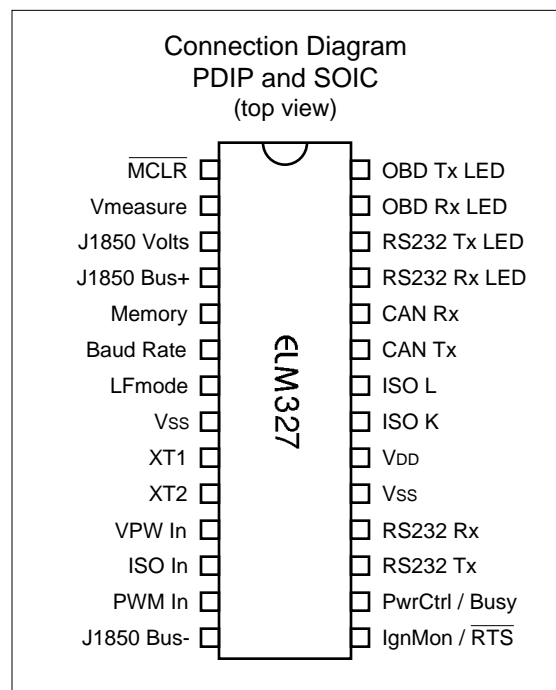
The following pages discuss all of the ELM327's features in detail, how to use it and configure it, as well as providing some background information on the protocols that are supported. There are also schematic diagrams and tips to help you to interface to microprocessors, construct a basic scan tool, and to use the low power mode.

Applications

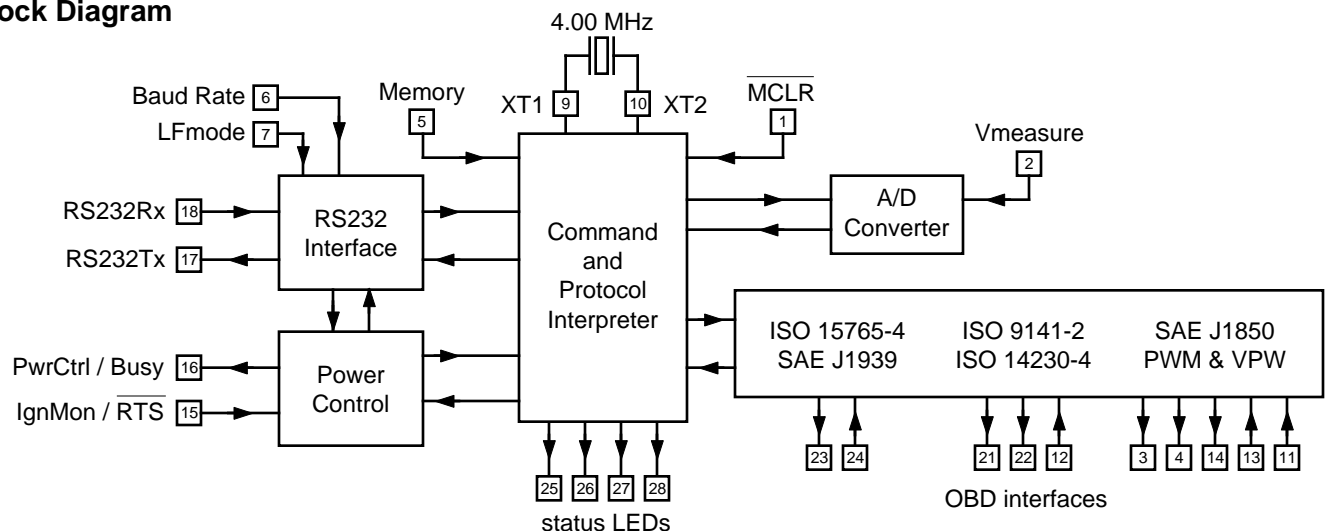
- Diagnostic trouble code readers
- Automotive scan tools
- Teaching aids

Features

- Power Control with standby mode
- Universal serial (RS232) interface
- Automatically searches for protocols
- Fully configurable with AT commands
- Low power CMOS design



Block Diagram





OBD Commands

If the bytes that you send to the ELM327 do not begin with the letters 'A' and 'T', they are assumed to be OBD commands for the vehicle. Each pair of ASCII bytes will be tested to ensure that they are valid hexadecimal digits, and will then be combined into data bytes for transmitting to the vehicle.

OBd commands are actually sent to the vehicle embedded in a data packet. Most standards require that three header bytes and an error checksum byte be included with every OBD message, and the ELM327 adds these extra bytes to your command bytes for you. The initial (default) values for these extra bytes are usually appropriate for most requests, but if you wish to change them, there is a mechanism to do so (see the 'Setting the Headers' section).

Most OBD commands are only one or two bytes in length, but some can be longer. The ELM327 will limit the number of bytes that can be sent to the maximum number allowed by the standards (usually seven bytes or 14 hexadecimal digits). Attempts to send more bytes will result in an error – the entire command is then ignored and a single question mark printed.

Hexadecimal digits are used for all of the data exchange with the ELM327 because it is the data format used most often in the OBD standards. Most mode request listings use hexadecimal notation, and it is the format most frequently used when results are shown. With a little practice, it should not be very difficult to deal in hex numbers, but some people may want to use a table such as Figure 1, or keep a calculator nearby. Dealing with the hex digits can not be avoided - eventually all users need to manipulate the results in some way (combining bytes and dividing by 4 to obtain rpm, dividing by 2 to obtain degrees of advance, converting temperatures, etc.).

As an example of sending a command to the vehicle, assume that A6 (or decimal 166) is the command that is required to be sent. In this case, the user would type the letter A, then the number 6, then would press the return key. These three characters would be sent to the ELM327 by way of the RS232 port. The ELM327 would store the characters as they are received, and when the third character (the carriage return) was received, would begin to assess the other two. It would see that they are both valid hex digits, and would convert them to a one byte value (the decimal value is 166). The header bytes and a checksum byte would then be added, and a total of five bytes would typically be sent to the vehicle. Note that the carriage return character is only a signal to the

ELM327, and is not sent to the vehicle.

After sending the command, the ELM327 listens on the OBD bus for replies, looking for ones that are directed to it. If a message address matches, the received bytes will be sent on the RS232 port to the user, while messages received that do not have matching addresses will be ignored (but are often still available for viewing with the AT BD command).

The ELM327 will continue to wait for messages addressed to it until there are none found in the time that was set by the AT ST command. As long as messages continue to be received, the ELM327 will continue to reset this timer, and look for more. Note that the IC will always respond to a request with some reply, even if it is to say 'NO DATA' (meaning that there were no messages found, or that some were found but they did not match the receive criteria).

Hexadecimal Number	Decimal Equivalent
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

Figure 1. Hex to Decimal Conversion



Talking to the Vehicle

The standards require that each OBD command or request that is sent to the vehicle must adhere to a set format. The first byte sent (known as the 'mode') describes the type of data being requested, while the second byte (and possibly a third or more) specifies the actual information that is required. The bytes which follow after the mode byte are known as the 'parameter identification' or PID number bytes. The modes and PIDs are described in detail in documents such as the SAE J1979, or ISO 15031-5 standards, and may also be defined by the vehicle manufacturers.

The SAE J1979 standard currently defines ten possible diagnostic test modes, which are:

- 01 - show current data
- 02 - show freeze frame data
- 03 - show diagnostic trouble codes
- 04 - clear trouble codes and stored values
- 05 - test results, oxygen sensors
- 06 - test results, non-continuously monitored
- 07 - show 'pending' trouble codes
- 08 - special control mode
- 09 - request vehicle information
- 0A - request permanent trouble codes

Vehicles are not required to support all of the modes, and within modes, they are not required to support all possible PIDs (some of the first OBDII vehicles only supported a very small number of them). Within each mode, PID 00 is reserved to show which PIDs are supported by that mode. Mode 01, PID 00 must be supported by all vehicles, and can be accessed as follows...

Ensure that your ELM327 interface is properly connected to the vehicle, and powered. Most vehicles will not respond without the ignition key in the ON position, so turn the ignition to on, but do not start the engine. If you have been experimenting, the state of your interface may be unknown, so reset it by sending:

```
>AT Z
```

You will see the interface leds flash, and then the IC should respond with 'ELM327 v2.1', followed by a prompt character. Now, you may choose a protocol that the ELM327 should connect with, but it is usually easier to simply select protocol '0' which tells the IC to search for one:

```
>AT SP 0
```

That's all that you need to do to prepare the

ELM327 for communicating with a vehicle. At the prompt, issue the mode 01 PID 00 command:

```
>01 00
```

The ELM327 should say that it is 'SEARCHING...' for a protocol, then it should print a series of numbers, similar to these:

```
41 00 BE 1F B8 10
```

The 41 in the above signifies a response from a mode 01 request (01 + 40 = 41), while the second number (00) repeats the PID number requested. A mode 02, request is answered with a 42, a mode 03 with a 43, etc. The next four bytes (BE, 1F, B8, and 10) represent the requested data, in this case a bit pattern showing the PIDs that are supported by this mode (1=supported, 0=not). Although this information is not very useful for the casual user, it does prove that the connection is working.

Another example requests the current engine coolant temperature (ECT). Coolant temperature is PID 05 of mode 01, and can be requested as follows:

```
>01 05
```

The response will be of the form:

```
41 05 7B
```

The 41 05 shows that this is a response to a mode 1 request for PID 05, while the 7B is the desired data. Converting the hexadecimal 7B to decimal, one gets $7 \times 16 + 11 = 123$. This represents the current temperature in degrees Celsius, but with the zero offset to allow for subzero temperatures. To convert to the actual coolant temperature, you need to subtract 40 from the value obtained. In this case, then, the coolant temperature is $123 - 40$ or 83°C .

A final example shows a request for the engine rpm. This is PID 0C of mode 01, so at the prompt type:

```
>01 0C
```

If the engine is running, the response might be:

```
41 0C 1A F8
```

The returned value (1A F8) is actually a two byte hex number that must be converted to a decimal value to be useful. Converting it, we get a value of 6904, which seems like a very high value for engine rpm.



Talking to the Vehicle (continued)

That is because rpm is sent in increments of 1/4 rpm! To convert to the actual engine speed, we need to divide the 6904 by 4. A value of 1726 rpm is much more reasonable.

Note that these examples asked the vehicle for information without regard for the type of OBD protocol that the vehicle uses. This is because the ELM327 takes care of all of the data formatting and translation for you. Unless you are going to do more advanced functions, there is really no need to know what the protocol is.

The above examples showed only a single line of response for each request, but the replies often consist of several separate messages, either from multiple ECUs responding, or from one ECU providing messages that need to be combined to form one response (see 'Multiline Responses' on page 42). In order to be adaptable to this variable number of responses, the ELM327 normally waits to see if any more are coming. If no response arrives within a certain time, it assumes that the ECU is finished. This same timer is also used when waiting for the first response, and if that never arrives, causes 'NO DATA' to be printed.

There is a way to speed up the retrieval of information, if you know how many responses will be sent. By telling the ELM327 how many lines of data to receive, it knows when it is finished, so does not have to go through the final timeout, waiting for data that is not coming. Simply add a single hex digit after the OBD request bytes - the value of the digit providing the maximum number of responses to obtain, and the ELM327 does the rest. For example, if you know that there is only one response coming for the engine temperature request that was previously discussed, you can send:

```
>01 05 1
```

and the ELM327 will return immediately after obtaining only one response. This may save a considerable amount of time, as the default time for the AT ST timer is 200 msec. (The ELM327 still sets the timer after sending the request, but that is only in case the single response does not arrive.)

Some protocols (like J1850 PWM) require an acknowledgement from the ELM327 for every message sent. If you provide a number for the responses that is too small, the ELM327 will return to the prompt too early, and you may cause bus

congestion while the ECU tries several times to resend the messages that were not acknowledged. For this reason, you must know how many responses to expect before using this feature.

As an example, consider a request for the vehicle identification number (VIN). This number is 17 digits long, and typically takes 5 lines of data to be represented. It is obtained with mode 09, PID 02, and should be requested with:

```
>09 02
```

or with:

```
>09 02 5
```

if you know that there are five lines of data coming. If you should mistakenly send 09 02 1, you might cause problems.

This ability to specify the number of responses was really added with the programmer in mind. An interface routine can determine how many responses to expect for a specific request, and then store that information for use with subsequent requests. That number can then be added to the requests and the response time can be optimized. For an individual trying to obtain a few trouble codes, the savings are not really worth the trouble, and it's easiest to just make a request, without regard to how many responses are expected.

We offer one additional warning when trying to optimize the speed at which you obtain information from vehicles. Prior to the APR2002 release of the J1979 standard, sending J1850 requests more frequently than every 100 msec was forbidden. With the APR2002 update, scan tools were allowed to send the next request without delay if it was determined that all the responses to the previous request had been received. Vehicles made prior to this time may not be able to tolerate requests at too fast a rate, so use caution with them.

Hopefully this has shown how typical requests are made using the ELM327. If you are looking for more information on modes and PIDs, it is available from the SAE (www.sae.org), from ISO (www.iso.org), or from various other sources on the web.