Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Caja negra en la nube para transporte por carretera

Autor: Juan Manuel Suárez Redondo

Tutor: Antonio Luque Estepa

Departamento de ingeniería electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Caja negra en la nube para transporte por carretera

Autor:
Juan Manuel Suárez Redondo

Tutor:
Antonio Luque Estepa
Profesor titular

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Proyecto Fin de Carrera: Caja negra en la nube para transporte por carretera

Autor: Juan Manuel Suárez Redondo

Tutor: Antonio Luque Estepa

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A Viki, por su capacidad para entenderme y su mente tan abierta.

Agradecimientos

En primer lugar, agradezco enormemente al profesor Doctor Don Antonio Luque Estepa la profesionalidad, la entrega y la dedicación personalizada con la que ha ejercido la labor de tutor de este proyecto. En segundo lugar, quisiera agradecer a la World Wide Web, en particular a sitios web como Google o Wikipedia las enormes posibilidades informacionales que ofrecen al ciudadano de a pie para acometer tareas o proyectos como el presente además de muchísimos otros, tanto en el ámbito profesional o académico como en el ámbito personal.

Juan Manuel Suárez Redondo

Autor del proyecto

Sevilla, 2017

Resumen

En el ámbito de la aviación moderna y actual, no tiene cabida el concepto de aeronave sin el concepto anexo de registrador de datos de vuelo o *caja negra*. Para aquella persona absolutamente profana en el ámbito de la ingeniería aeroespacial, dentro de este contexto, un registrador de datos de vuelo o *caja negra* no es otra cosa que un dispositivo estanco, robusto, a prueba de golpes, a prueba de fuego, etc. de registro continuo de los datos relativos al funcionamiento y al trayecto de la aeronave, para que en caso de que desafortunadamente tenga lugar un accidente de aviación, puedan recabarse los motivos técnicos precisos que lo originaron.

El objetivo de este proyecto también es implementar una *caja negra*, solo que en vez de a nivel hardware, se hará a nivel software (en forma de app para el sistema operativo Android) y en vez de destinada al uso en vehículos de transporte aéreo se hará destinada al uso en vehículos de transporte por carretera.

A lo largo del apartado concerniente al estado del arte se hará una breve explicación de los dispositivos hardware y/o software que ya existen y que funcionan (total o parcialmente) como *cajas negras* para vehículos de transporte por carretera, puesto que como el lector podrá observar conforme vaya leyendo el mencionado capítulo, el presente proyecto no es pionero en su ámbito, aunque sí único dentro de este.

Abstract

This final college career project is all about a software implementation of a black box for terrestrial vehicles through OBD2 interface. The operative system choosed has been Android and the IDE used has been Intel XDK together with Apache Cordova. It is also possible to recompile the source code (with some adaptations) for the use with other mobile operative systems like iOS or Windows Phone.

ÍNDICE

1 INTRODUCCIÓN Y OBJETIVOS

Si hubiera una tercera guerra mundial, las armas de la cuarta serían palos y piedras.

- Albert Einstein -

Los registradores de datos de vuelo, más comunmente conocidos como *cajas negras*, son un elemento de la era actual de la aviación (tanto civil como militar) del que sería inconcebible que carecieran las actuales aeronaves, sea cual sea el propósito para el que hayan sido diseñadas, fabricadas o concebidas. Se denomina *caja negra* o registrador de datos de vuelo al dispositivo que registra la actividad de los instrumentos y las conversaciones en la cabina de la aeronave. Su función es almacenar información que, en caso de que desafortunadamente tenga lugar un accidente de aviación, permita analizar lo ocurrido en los momentos previos a la ocurrencia del suceso.

Los primeros registradores de datos de vuelo se empezaron a usar a finales de los años 1950 y se les llamó, como se ha mencionado anteriormente, *cajas negras*. Esta denominación perduró incluso después de que se empezasen a pintar de color naranja con el objetivo de facilitar su localización tras un accidente. El origen de la denominación *caja negra* no está claro: algunos prototipos estaban pintados de negro y otros eran cámaras oscuras con placas fotográficas, aunque en lo que respecta a su/s entrada/s y salida/s todos ellos se comportan como *cajas negras* desde el punto de vista de la ingeniería de sistemas.

Figura 1: registrador de datos de vuelo o *caja negra* de una aeronave.

El objetivo que persigue el presente proyecto fin de carrera es implementar una *caja negra*, solo que en vez de ser a nivel hardware y permanecer en el interior de un compartimento estanco, robusto, a prueba de golpes, impermeable, a prueba de fuego, etc., se implementará a nivel software a modo de aplicación o app para el sistema operativo Android. Además, esta *caja negra* estará destinada al uso en vehículos de transporte por carretera que dispongan de interfaz OBD2 en vez de al uso en vehículos de transporte aéreo. A efectos prácticos, la gran mayoría de los vehículos cuyo año de fabricación sea 1996 o posterior dispondrán de interfaz OBD2. En el siguiente apartado se detallan los objetivos de desarrollo concretos que deberán alcanzarse.

1.1.-Objetivos planteados.

Los objetivos que se han marcado de cara al desarrollo de la app del presente proyecto fin de carrera y que esta forzosamente deberá cumplir son los siguientes:

- Desarrollar una app que sea capaz de obtener la información relativa al funcionamiento del motor del vehículo al que esté vinculado o conectado y mostrarla por pantalla.
- Desarrollar una app que sea capaz de obtener la información relativa a la ubicación o localización y a la velocidad del terminal Android e igualmente mostrarla por pantalla.
- Desarrollar una app que sea capaz de enviar los parámetros que se obtengan tanto del terminal Android como del vehículo al que se encuentre conectado a un servidor de base datos ubicado en *la nube* con una periodicidad temporal configurable por el usuario.

Si la app cumple estos tres objetivos, el desarrollo de la misma podría darse teóricamente por finalizado a falta de la realización de diversas pruebas de funcionamiento que descarten cualquier tipo de error que inicialmente se haya pasado por alto y que afecten a la correcta utilización de la misma por parte del usuario.

1.2.-Necesidad y justificación.

En la actualidad ya existen diversas soluciones de *cajas negras* a nivel hardware para vehículos de transporte por carretera que casi cualquier usuario con conocimientos básicos o intermedios en informática y electrónica puede implementar para su uso personal, motivo por el cual no están disponibles comercialmente, sino que son proyectos o soluciones DIY (Do It Yourself o hágalo usted mismo). Además, los autores de este tipo de soluciones de *cajas negras* normalmente suelen no hacerse responsables de un posible malfuncionamiento que el usuario final pueda experimentar durante el uso del dispositivo, aún cuando éste haya seguido al pie de la letra todos y cada uno de los pasos que el autor explica detalladamente en su web o blog...

Por otra parte, también existen soluciones que implementan total o parcialmente a nivel software a modo de apps para el sistema operativo Android la funcionalidad de una *caja negra*, como por ejemplo la app *Torque Lite*. Sin embargo, esta app, aún teniendo muchísimas funcionalidades, carece de la principal funcionalidad que sí tiene la app del presente proyecto fin de carrera: la integración con la nube, es decir, la posibilidad de enviar los datos que se obtengan tanto del vehículo como del propio terminal Android a una base de datos ubicada en un servidor remoto y totalmente externa al vehículo y al propio terminal Android.

Por tanto, la app del presente proyecto fin de carrera será capaz, en primer lugar, de obtener los parámetros de funcionamiento del vehículo al que esté conectado a través de su interfaz Bluetooth; en segundo lugar, será capaz de obtener los parámetros relativos a la ubicación del propio terminal Android en el que se esté ejecutando; y en tercer y último lugar, la app será capaz de enviar la información obtenida del vehículo y del propio terminal Android a un servidor externo de base de datos en la nube previamente configurado por el usuario.

Llegados a este punto, cabría plantearse algunas cuestiones, como por ejemplo las siguientes:

- ¿Por qué desarrollar una app para obtener los parámetros de funcionamiento del motor de un vehículo de transporte por carretera? ¿Es esto realmente importante? ¿Acaso no existen ya otras apps que implementan esta funcionalidad?
- ¿Por qué enviar la información que se obtenga a un servidor externo de base de datos en la nube? ¿Acaso no es suficiente con mostrarla en pantalla o como mucho almacenarla en la memoria interna del terminal?

- ¿Por qué incluir también parámetros obtenidos del propio terminal Android? ¿No se usan ya otras apps como por ejemplo la app de Google Maps para este fin?

En primer lugar, a decir verdad, no es totalmente imprescindible obtener los parámetros de funcionamiento del motor de un vehículo de transporte por carretera. De hecho, la inmensa mayoría de los vehículos que circulan actualmente lo hacen sin que se obtenga ni se registre esta información. Sin embargo, desde el año 1996 aproximadamente, todos los vehículos de nueva fabricación disponen de interfaz OBD2, la cual posibilita la lectura de estos parámetros. Al igual que en el caso del transporte aéreo, en caso de que tenga lugar un desafortunado accidente de circulación, la información obtenida del vehículo por la app servirá para determinar lo ocurrido instantes antes de que este tuviera lugar. Esta información podría ser cotejada por las autoridades junto con el resto de indicios o pruebas del accidente de circulación para determinar la responsabilidad del mismo. Como se ha comentado anteriormente no es totalmente necesario leer estos parámetros de funcionamiento, pero la interfaz OBD2 facilita enormemente las cosas, dado que solamente hay que preocuparse de conectar al vehículo un dispositivo Bluetooth apropiado con interfaz OBD2: no hay que llevar a cabo ningún cambio en el vehículo como por ejemplo instalar sensores, un sistema de adquisición de datos, un microcontrolador, etc. por lo que además esta solución o implementación de *caja negra* para transporte por carretera podría también considerarse *low cost* o de bajo coste. Además, la gran mayoría de las apps que existen actualmente para obtener los parámetros de funcionamiento del motor de un vehículo en tiempo real (entre las que se incluye la anteriormente mencionada *Torque Lite*) no implementan la funcionalidad de registro de los datos que se obtienen, es decir, que simplemente van mostrando los valores de los parámetros de funcionamiento del motor del vehículo en tiempo real según estos se van obteniendo, pero no sería posible analizar sus valores durante un determinado intervalo de tiempo concreto (por ejemplo justo antes de que haya tenido lugar un accidente de circulación) ya que no se ha llevado a cabo un registro de los mismos en ninguna parte.

En segundo lugar, en lo que respecta a enviar la información obtenida a un servidor remoto de base de datos en la nube, comentar que la razón de ser de esta funcionalidad no es otra que prevenir la pérdida de datos que tendría lugar en caso de que el accidente de circulación sea desastroso o fatal desde el punto de vista material o físico. Si el vehículo o el terminal han sufrido daños físicos graves y la información que pudiera ayudar a esclarecer lo ocurrido instantes antes de que el accidente de circulación tuviera lugar estaba en ellos, lamentablemente es muy posible que no se pueda recuperar. Sin embargo, si la app ha ido enviando periódicamente la información que ha ido obteniendo a un servidor externo de base de datos en la nube, incluso aún en el caso de que el terminal Android, el vehículo y todos sus ocupantes desaparecieran y nunca más se volviera a saber de ellos, todavía sería posible recuperar la información del servidor de base de datos en la nube, ya que este es totalmente externo y ajeno tanto al vehículo como al propio terminal Android.

Y en tercer y último lugar, si nos preguntamos por qué sería útil también incluir en la información enviada al servidor de base de datos en la nube parámetros obtenidos del propio terminal Android (básicamente su ubicación) si ya existen otras apps que son capaces de obtener esta información, comentar que la información relativa a la ubicación del terminal, por si sola, es cierto que ya son capaces de mostrarla y/o de registrarla otras apps como por ejemplo la app de Google Maps. Sin embargo, aún no exite ninguna app para el sistema operativo Android que sea capaz de combinar la información obtenida del vehículo a través de su interfaz OBD2 con la información relativa a su ubicación o localización. La app del presente proyecto fin de carrera es capaz de combinar ambos tipos de información en un único registro de la base de datos en la nube, de manera que en caso de que desafortunadamente tenga lugar un accidente de circulación, a las autoridades responsables de determinar el origen, la causa, la autoría del mismo, etc. esta información les será de mucha utilidad porque les permitirá saber, por cada registro que se haya hecho en la base de datos en la nube, además de la fecha y la hora, cuales eran la localización o ubicación y la velocidad del terminal (y por tanto del vehículo), cuáles eran las revoluciones por minuto (RPM) del motor del vehículo, cuál era la velocidad del vehículo en ese determinado instante, la posición del acelerador (en un intervalo de 0 a 100), la temperatura del motor, la cantidad de aire que atraviesa el motor del vehículo, es decir, el flujo de masa de aire, etc... De forma separada, la información relativa a la ubicación o localización del vehículo, la relativa al funcionamiento de su motor y la relativa a la fecha y la hora en la que se han obtenido estos parámetros siguen siendo útiles, pero si se combinan entre ellas el tiempo que los expertos oficiales en accidentes de circulación necesitan para hacer su labor puede ser menor, las conclusiones a las que estos lleguen pueden ser más determinantes que sin el uso de la app e incluso, por qué no, se podrían resolver aquellos casos de accidentes de circulación dudosos (si los hubiera) en los que la falta de indicios o la incoherencia entre los testimonios de los afectados pudieran dar lugar a inconsistencias en los informes periciales.

2 ESTADO DEL ARTE

Solo somos una especie evolucionada de mono que habita un planeta de tamaño pequeño que orbita alrededor de una estrella mediana.

- Stephen Hawking -

El estado de actual de desarrollo de dispositivos hardware y/o software cuya finalidad específica es la obtención de datos relativos al funcionamiento de vehículos terrestres no ferroviarios es muy amplio. Por una parte, primeramente, habría que mencionar el tacógrafo, tanto en su versión analógica más duradera o tradicional como en su relativamente nueva y reinventada versión digital. Por otra parte, tampoco hay que olvidar mencionar todos aquellos intentos, exitosos o no a nivel de número de usuarios, de implementación de dispositivos cuya finalidad, al igual que la del presente proyecto, no es otra que la obtención de datos de funcionamiento del vehículo en tiempo de conducción.

A nivel de ingeniería "extraoficial" se pueden encontrar en Internet numerosos proyectos de implementación de este tipo de sistemas, dado que, al menos por ahora, a excepción del sistema EDR (Event Data Recorder) y del tacógrafo (digital o analógico), aún no existe ninguna solución de recopilación de información de funcionamiento del vehículo en tiempo de conducción con amplia acepctación por parte de fabricantes y/o usuarios.

2.1.-El tacógrafo [1], [2].

El término tacógrafo proviene de los términos de origen griego 'takhos' y 'grafos', que respectivamente significan rapidez y escribir. Un tacógrafo no es otra cosa que un dispositivo electrónico que registra diversos sucesos o eventos originados en un vehículo de transporte terrestre durante su conducción, ya sea este de carga o de pasajeros, y ferroviario o de carretera. Los eventos o sucesos que normalmente es capaz de registrar un tacógrafo suelen ser típicamente la distancia recorrida por el vehículo, la velocidad promedio y la velocidad máxima, las aceleraciones y frenadas bruscas, los periodos de tiempo durante los cuales el vehículo permanece detenido con el motor en marcha, los tiempos de descanso del conductor, etc.

Estos datos, dependiendo del tipo de implementación del tacógrafo (analógica o digital), pueden ser recopilados en una computadora y almacenados en una base de datos o también imprimirse gráficamente en discos de papel para su posterior análisis e interpretación (ver figura 2).

Figura 2: leyenda de interpretación de las marcas imprimidas por un tacógrafo analógico.

2.2.-El tacógrafo digital [3].

El relativamente nuevo tacógrafo digital posee una apariencia similar a la del tradicional tacógrafo analógico. El sistema se compone de una unidad de vehículo, un sensor de velocidad o de distancia y una serie de tarjetas inteligentes que sustituyen a los discos de papel de su predecesor analógico. Estas tarjetas inteligentes disponen de un chip que almacena la información de conducción al mismo tiempo que da acceso a determinadas funciones según cual sea el perfil del usuario (conductor del vehículo, empresa propietaria del vehículo, autoridades de control o taller de reparaciones). La información que queda almacenada en el tacógrafo digital es exactamente la misma que la que almacena su predecesor analógico en cuanto a tiempos y velocidades, solo que en este caso el nivel de seguridad que ofrece el tacógrafo digital es tan alto que se garantiza que la información registrada será prácticamente imposible de manipular.

En cuanto al a unidad de vehículo anteriormente mencionada, ésta se encuentra instalada en el interior del mismo, concretamente dentro de la cabina del conductor, de forma que este pueda interoperar con él. La unidad de vehículo se comunicará con el sensor, que normalmente estará instalado en la caja de cambios, encontrándose ambos elementos conectados entre sí mediante un cable.

La información recopilada por el tacógrafo digital relativa al uso del vehículo se almacena en dos tipos de memorias: la de la unidad intravehicular y la de las tarjetas inteligentes.

2.3.-Dispositivo de monitorización de vehículo mediante microcontrolador 16F886 [4].

Este dispositivo no es más que otra solución OBD2 para monitorizar los sensores de un vehículo. Soporta el protocolo ISO 9141-2 o ISO 14230-4 (también conocido como “Keyword Protocol 2000” o KWP). Ambos estándares ISO son casi idénticos. La ventaja característica de esta solución sobre otras es que es low-cost o de bajo coste en el sentido de que no necesita un Arduino, una Raspberry Pi o un teléfono inteligente para poder llevarla a la práctica: su coste total estimado es de alrededor de 10€.

Elementos necesarios para construirlo:

- Un conector OBD2 macho de 16 pines.
- Un microcontrolador Microchip 16F886.
- Una pantalla LCD HD44780 o compatible.
- Un transistor BC547 o un transistor 2N3940.
- Un regulador LM7805 de 1A.
- Un condensador cerámico de 330nF.
- Un condensador cerámico de 100nF.
- Una resistencia de 510Ω.

- Una resistencia de 2200Ω.
- Una resistencia de 47KΩ.
- Una resistencia de 33KΩ.
- Una resistencia de 100Ω (opcional para un LED azul). Para LEDs de otros colores hay que usar una resistencia de diferente valor.
- Un LED azul (opcional).
- Un zumbador de 5V y menos de 20mA (opcional).

Principales funcionalidades:

- Muestra la carga del motor del vehículo, la temperatura, la velocidad y las revoluciones por minuto en una pantalla LCD retroiluminada.
- Muestra también de forma óptica la carga del motor del vehículo mediante un LED que cambia de intensidad gracias al modulo PWM del microcontrolador 16F886.
- Si la temperatura supera los 93 grados se activa un zumbador que avisa acusticamente de este hecho.
- Dispone "watchdog" que detecta problemas de comunicación entre el microcontrolador y la *ECU* (Engine Control Unit o unidad de control del motor) del vehículo.

Funcionalidades que han quedado pendientes de ser implementadas:

- Mostrar códigos de error de los errores detectados en el vehículo.
- Limpiar estos códigos de error.
- Inicio rápido.

Este dispositivo ha sido probado en un Citroën C1 gasolina del año 2013 y en un Volkswagen Touran gasolina del año 2003. También funcionaría probablemente en un Peugeot 107 y en un Toyota Aygo del año 2013, ya que ambos vehículos son técnicamente idénticos en lo que respecta a la interfaz OBD2. Sin embargo, hay muchas variantes de los protocolos ISO 9141-2 e ISO 14230-4, por lo que existe la posibilidad de que no sea interoperable con absolutamente todos los vehículos con interfaz OBD2, de manera que puede que en algunos casos sea necesario hacer algunos cambios para lograr que funcione. De todas formas, en [14] se puede comprobar la compatibilidad de un cierto modelo de vehículo con estos estándares.

2.4.-Caja negra con dos cámaras: Car Angel BBX2 [5].

El dispositivo "Car Angel BBX2" es una caja negra con grabación de vídeo mediante doble cámara de alta calidad, que incorpora un tacógrafo por GPS para control de rutas y evaluación de la conducción, conexión Wi-Fi y una pantalla LCD para facilitar la interacción con el usuario.

El aparato permite evaluar la calidad de la conducción en periodos muy largos gracias a su capacidad para almacenar grandes cantidades de información. Esta funcionalidad resulta especialmente relevante para las compañías aseguradoras y también en todo tipo de casos en los que un vehículo de un cierto valor es puesto en manos de terceros. También permite identificar oportunidades de reducción de consumo de combustible mediante la mejora de la conducción, pudiéndose alcanzar ahorros en el consumo de combustible de hasta el 20%.

Cuenta con la funcionalidad EVDR (Event Video Data Recording o grabación de eventos en vídeo digital), una especie de función de caja negra con vídeo, que graba en vídeo de alta calidad 15 segundos antes y después de un evento dado, haya sido este disparado o activado a voluntad del conductor o bien por una aceleración o deceleración brusca.

Permite la grabación manual, en el sentido de que cuando una sacudida del vehículo no es lo suficientemente brusca como para activar la alarma del acelerómetro, el conductor puede grabar lo que está ocurriendo a voluntad pulsando el botón REC durante 15 segundos.

Por último, el tacógrafo digital por GPS le permitirá conocer el uso que se le dio al vehículo durante periodos largos: velocidad, posición en Google Maps, simulación dinámica de rutas, velocidades máximas superadas, etc…

2.5.-EDR: Event Data Recorder o grabador de datos de eventos [6].

Un dispositivo EDR (Event Data Recorder o grabador de datos de eventos) es un dispositivo instalado en algunos automóviles para grabar información relativa a accidentes de circulación o choques entre vehículos. En camiones diesel modernos, los eventos de grabación de datos se disparan electrónicamente por la detección de problemas (a menudo llamados fallos) en el motor del vehículo o también por un cambio repentino en la velocidad del vehículo. Una o más de una de estas condiciones puede originarse como consecuencia de un accidente de circulación. La información de los dispositivos EDR puede ser recopilada y analizada después de un choque para determinar qué estaba haciendo el vehículo antes, durante y después del choque o evento.

2.5.1.-Funcionamiento de un EDR.

Existen muchas patentes diferentes relativas a varios tipos de funcionalidades de estos dispositivos. Algunos dispositivos EDR están continuamente almacenando datos, sobreescribiendo los minutos previos antes de que un choque detenga su funcionamiento, mientras que otros empiezan a funcionar mediante eventos que sean detectados (fidedigna o erróneamente) como un choque, como por ejemplo por cambios repentinos y bruscos en la velocidad, y pueden continuar grabando hasta que el accidente acabe o hasta que el tiempo de grabación se agote.

Los dispositivos EDRs pueden grabar un amplio rango de elementos, entre los cuales se incluyen la activación de los frenos, la velocidad en el momento del impacto, el ángulo del volante y si los cinturones de seguridad estaban abrochados o no abrochados en el momento del accidente. Los actuales dispositivos EDR guardan la información internamente en una EEPROM hasta que ésta sea extraída del sistema. Algunos vehículos tienen sistemas de comunicación que son capaces de transmitir algunos datos (como por ejemplo una alerta que indique que los airbags se han disparado) a una ubicación remota.

La mayoría de los dispositivos EDR en automóviles y camiones ligeros son parte del módulo de control del sistema de control, que monitoriza aceleraciones o deceleraciones bruscas y determina que controles hay que aplicar (airbag y/o tensores de los cinturones) hay que aplicar. Después de que las decisiones sobre los controles que hay que aplicar se hayan tomado, y si aún hay energía disponible, los datos son escritos en la memoria. Los datos descargados de dispositivos EDR antiguos usualmente contiene entre 6 y 8 páginas de información, mientras que muchos sistemas más modernos incluyen muchos otros elementos de datos y requieren más páginas de información, dependiendo de las circunstancias de las colisiones que hayan tenido lugar y del tiempo transcurrido entre ellas, entre otros factores.

También es posible que no se pueda recuperar ningún dato de un dispositivo EDR. Una de estas situaciones puede tener lugar durante una pérdida catastrófica de energía eléctrica a comienzo de un evento de colisión. En esta situación, la reserva de energía de los condensadores del módulo de control del sistema de control puede haber sido completamente gastada por el despliegue de los airbags, no quedando energía suficiente para poder escribir datos en la EEPROM.

Por otra parte, la mayoría de los dispositivos EDR en camiones pesados son parte del módulo de control electrónico del motor, que controla la temporización de la inyección de carburante además de otras funciones en motores diesel modernos de alta potencia.

Las funciones de los dispositivos EDR son diferentes para cada fabricante de motores, aunque la mayoría reconoce eventos relativos al motor tales como paradas repentinas baja presión de carburante o pérdida de líquido refrigerante. Los motores de las marcas Detroit Diesel, Caterpillar, Mercedes-Benz, Mack Trucks y Cummings se encuentras entre los que disponen de esta funcionalidad. Cuando un evento relacionado con un fallo tiene lugar, los datos relacionados con este se escriben en memoria. Cuando un generado por una reducción de la velocidad del vehículo tiene lugar, los datos que se escriben en la memoria pueden incluir casi dos minutos de datos sobre la velocidad del vehículo, la activación de los frenos y la activación del embrague. Los datos pueden ser descargados posteriormente usando los cables y el software específicos del motor en cuestión. Estas herramientas de software específicas de cada marca fabricante de motores permiten a menudo la monitorización de las horas de servicio del conductor, el gasto de combustible, los tiempos en los que el motor ha estado sin utilizar, la velocidad media y otra información relativa al mantenimiento y al funcionamiento del vehículo.

Por último, algunos EDRs solo guardan el seguimiento de la velocidad del vehículo a lo largo de su longitud, pero no de forma lateral. Los analistas de accidentes de circulación generalmente se centran en el momento en

el que tuvo lugar la colición, en su energía y en el daño producido por esta, y posteriormente comparan sus estimaciones de velocidad con la velocidad registrada por el dispositivo EDR para crear una vista completa del accidente.

2.5.2.-Implementación de dispositivos EDRs.

Los EDRs se introdujeron en la temporada de 1993 del campeonato americano CART y también en el campeonato mundial de fórmula 1 de 1997. Esto permitió estudiar accidentes o colisiones entre vehículos que permitieron a su vez desarrollar nuevas normas o estándares de fabricación y adoptar medidas de seguridad para reducir los daños ocasionados por estos.

El uso de los EDRs en vehículos de carretera varía ampliamente de fabricante en fabricante. General Motors y Ford implementan la tecnología en la mayoría de sus modelos recientes, mientras que por ejemplo Mercedes-Benz y Audi no usan ningún tipo de EDR. Alrededor del año 2003 había alrededor de 40 millones de vehículos equipados con EDRs. En Reino Unido, muchos vehículos policiales y de emergencias son equipados con una versión más precisa y detallada que es producida por una de entre varias compañías independientes entre sí. También la policía londinense es usuaria a largo plazo de los EDRs y han usado los datos recuparados tras un accidente para condenar o declarar culpables tanto a oficiales de policía como a ciudadanos comunes.

2.6.-Black Box Pi: caja negra implementada con una Raspberry Pi [7].

Black Box Pi es una caja negra para coches de código abierto implementada con ayuda de una Raspberry Pi junto con otro hardware de bajo coste combinados con almacenamiento en la nube de los datos disponibles. La Raspberry Pi actúa a modo de concentrador y obtiene datos relevantes del vehículo mientras este se encuentra en funcionamiento. Esta información se almacena en la nube y es accesible desde cualquier dispositivo con conexión a Internet (ver figura 3).

Por menos de 100 dólares de gasto en hardware y unas cuantas horas de instalación se puede conseguir una caja negra para coches plenamente funcional. Todos los datos que genera el dispositivo pueden subirse al servidor propiedad del usuario, de forma que éste tiene pleno control sobre los datos.

El dispositivo recopila información de forma continua durante todo el tiempo de funcionamiento del vehículo y registra o almacena tantos datos como sean posibles. Entre los tipos datos que recopila el vehículo se encuentran los datos obtenidos de la interfaz OBD2, los datos obtenidos del GPS, la temperatura, la aceleración, la orientación dada por la brújula, etc…

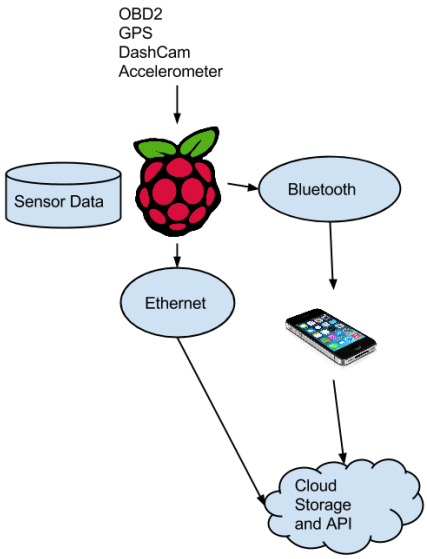

Figura 3: esquema de funcionamiento de la Black Box Pi.

Hasta ahora todo lo que hay desarrollado relativo a la interfaz OBD2 se puede dividir en dos categorías:

- Monitorización en tiempo real de los parámetros de funcionamiento del vehículo.
- Ajuste de la *ECU* (Engine Control Unit o unidad de control del vehículo) para obtener más potencia del vehículo.

Sin embargo, este nuevo dispositivo consigue un nuevo tercer tipo de uso de la interfaz OBD2: la monitorización a largo plazo de todos los datos disponibles para su posterior análisis.

Aunque ya existen aplicaciones para smartphone que se pueden conectar a un lector OBD2 con Bluetooth para extraer los datos, esto no es una solución ideal para el registro continuo de la información proporcionada por el vehículo, ya que para este propósito es necesario un dispositivo dedicado (un dispositivo que únicamente se encargue de esta tarea y de ninguna otra) que se encuentre encendido siempre que el vehículo se encuentre en funcionamiento y que esté continuamente registrando la información proporcionada por el vehículo.

Con el objetivo de almacenar en la nube toda esta información, es necesario implementar por software una arquitectura específica de servidor de almacenamiento de datos, que comprende de los siguientes elementos:

- Base de datos MySQL.
- API de servicios web JSON + REST.
- SDK (Software Development Kit o kit de desarrollo de software) que el software OBD2 usará para almacenar los datos que está capturando.
- Software libre de captura de datos a través de la interfaz OBD2 que se ejecutará en segundo plano en la Raspberry Pi.
- Buffer local para el almacenamiento temporal de información que será sincronizado con el servicio de almacenamiento de datos en la nube cuando haya conexión a Internet disponible.

2.7.-App CaroO Free [8].

CaroO Free es una app que soporta un paquete completo de características entre las cuales se la detección de colisión por vídeo, el registro de la ruta de conducción con función GPS, así como la monitorización y el diagnóstico del vehículo mediante el uso de un dispositivo OBD2. Las funciones de grabación de vídeo activada por eventos y la de monitorización del vehículo pueden usarse juntas o por separado. Además, el funcionamiento en segundo plano es muy estable.

Otras características de la app son las siguientes:

- Soporte para grabación y control de la calidad de vídeo de hasta 1920x1080 (full HD).
- Control del enfoque y de la exposición de la cámara (dependiendo de las capacidades del terminal).
- Borrado automático de archivo para el modo de grabación normal o de emergencia.
- Aviso de emergencia mediante por SMS, llamada y vídeo de Youtube.
- Monitorización del vehículo mediante la conexión a un dispositivo compatible con el integrado ELM327 con interfaz OBD2.
- Monitorización de parámetros como las revoluciones por minuto, la velocidad, el voltaje de la batería, la temperatura del refrigerante, la presión del combustible y muchos otros datos dependiendo del modelo del vehículo.
- Aviso ante corte del suministro de combustible, ante frenada o aceleración brusca, información sobre el grado de ecología de la conducción en curso e indicador de inactividad.
- Opción de inicio y finalización automáticas.
- Interoperatibilidad con dispositivos Android Wear para control y notificaciones.

Se recomienda el uso de terminales con versiones de Android 4.0 o superiores, aunque la mayoría de los terminales con versiones de Android 2.2 o superiores también son soportados.

2.8.-Rastreador de vehículo por GPS y caja negra por interface OBD2 AX5 [9].

El rastreador de vehículo AX5 es un dispositivo de seguimiento por GPS y una caja negra que se acopla al puerto OBD2 de cualquier vehículo. Su instalación es “plug and play”, es decir, que no necesita que el usuario tenga que configurarlo, sino que puede empezar a usarlo justo tras enchufarlo a la interfaz OBD2 del vehículo.

Además, el dispositivo AX5 usa la red GSM para transmitir información sobre la ubicación a través de redes 3G.

Para usar el dispositivo, hay que hacer login en la aplicación web desde cualquier dispositivo conectado a Internet para ver la ubicación de los vehículos equipados con el dispositivo. La aplicación web permite monitorizar multiples vehículos al mismo tiempo en paneles diferentes. También se pueden obtener alertas por email o SMS relativas a la velocidad de los vehículos, el arranque o parada de los mismos y los tiempos de inactividad.

Otra funcionalidad bastante curiosa es la de vallado o cercado geográfico, que permite configurar un vallado virtual alrededor de áreas específicas, de manera que se reciba una alerta cuando un determinado vehículo entra o sale de esa área. Por último, destacar también la función de playback, que permite reproducir una y otra vez una determinada trayectoria que ha seguido un determinado vehículo, aunque este ya se encuentre realizando otra ruta.

Sin embargo, no todo son ventajas, ya que el rastreo por GPS de el/los vehículo/s requiere de una subscripción mensual, aunque el primer mes de subscripción es gratuito. Después del primer mes de subscripción gratuito, los planes de subscripción tienen un coste desde 14,95 dólares, pero no hay compromiso de permanencia: se puede solicitar la baja en cualquier momento, además de que no hay cuota de activación del dispositivo.

2.9.-Caja negra grabadora EQP-104 OBD 11 [10].

Esta grabadora de datos todo en uno ultracompacta resulta extremadamente útil para diagnosticar o detectar fallos intermitentes de los componentes eléctricos del vehículo en vehículos que soporten el sistema OBD 11. Permite la gestión del consumo de combustible tanto por parte del vehículo como por parte del conductor, el análisis del estilo de conducción de cada conductor que use el vehículo, así como el registro de información relativa al funcionamiento del motor.

Tiene capacidad para almacenar hasta 300 horas de registro de datos. Los parámetros que se pueden registrar son la fecha y la hora actuales, las revoluciones por minuto del motor, la posición del acelerador y muchas más que dependen del modelo específico de vehículo al que se conecte.

3 DESARROLLO

No me interesan la alabanza o la crítica, me basta con seguir mis propios sentimientos.

- Wolfgang Amadeus Mozart -

El desarrollo íntegro de la app se ha llevado a cabo usando el software Intel XDK [11]. Su funcionamiento se rige en torno a la creación y realización de proyectos, por lo que para la realización de la app del presente proyecto fin de carrera se ha creado un nuevo proyecto que inicialmente ya venía con la estructura de archivos básica predeterminada. El funcionamiento del software Intel XDK va de la mano con el de Apache Cordova [12], de manera que el resultado final de cada proyecto es una app que puede compilarse para múltiples plataformas objetivo (Android, iOS, Windows Phone…) usando un único código fuente de origen.

3.1.-Estructura.

Se describe en este apartado, en primer lugar, la estructura de funcionamiento de la app; en segundo lugar, la estructura de directorios y ficheros creada inicialmente por el entorno de desarrollo Intel XDK; y en tercer y último lugar, la estructura del código fuente de la app.

3.1.1.-Estructura de funcionamiento de la app.

El funcionamiento de la app se estructura básicamente en torno a la función *onDeviceReady*, que será invocada una vez que la app pase a estado *ready* o listo. Dentro de esta función se definen la gran mayoría de funciones, variables y constantes que se usarán en durante el transcurso de la ejecución de la app, por lo que no estarán disponibles fuera del ámbito de influencia de esta función.

Una vez que se llama a la función *onDeviceReady*, se declaran o definen practicamente toda la totalidad de las funciones, constantes y variables de la app (a excepción de unas pocas) y a continuación se comienzan hacer las primeras llamadas a diferentes funciones de los plugins integrados en el proyecto creado ad hoc para el desarrollo de la app en el entorno de desarrollo Intel XDK.

Lo primero que la función *onDeviceReady* ejecuta es una llamada a la función *cordova.plugins.diagnostic.isLocationEnabled*, que determinará si está disponible el acceso a la ubicación o localización del terminal Android en el que se está ejecutando la app.

A continuación, se invoca a la función *bluetoothSerial.isEnabled*, que determinará análogamente a la invocación de la función *cordova.plugins.diagnostic.isLocationEnabled* si la interfaz Bluetooth del terminal está habilitada (y por tanto disponible para poder usarla).

Lo siguiente que ejecuta la función *onDeviceReady* son llamadas a varias funciones del plugin *BackgroundMode* con vistas a permitir que la app pueda ejecutarse en segundo plano exactamente igual que cuando lo haría en primer plano. También se invoca a otra función de este mismo plugin cuyo objetivo es impedir que la app se cierre cuando se pulse el botón *back* del terminal Android.

Después de esta serie de llamadas a diferentes funciones de distintos plugins integrados previamente en la app,

se programa mediante la función *setInterval* de Javascript la invocación cada medio segundo de la función *comprobar_validacion_o_detencion_configuracion*. Esta función, como su propio nombre indica, comprobará cada vez que se la invoque (cada medio segundo) si se ha pulsado el botón *VALIDAR* o *DETENER*, y evidentemente actuará en consecuencia.

A continuación, se intentan recuperar los últimos valores disponibles de los parámetros de configuración de conexión con la base de datos en la nube con vistas a no obligar al usuario de la app a tener que introducirlos de nuevo en caso de que estos no hayan cambiado desde la última vez que se guardaron.

Tras haber recuperado, dentro de lo posible, los últimos valores disponibles de los parámetros de configuración de conexión con la base de datos en la nube, se invoca a la función *comprueba_conexion_Internet*, que como su propio nombre indica comprobará si el terminal tiene disponible una conexión a Internet válida.

Justo después de conocer el estado de conectividad a Internet del terminal (y solo en caso de que el terminal no disponga de una conexión válida a Internet), se lanza un mensaje de advertencia con dos posibles opciones que anima al usuario bien a cerrar la app en caso de que prefiera no seguir con su ejecución sin conexión a Internet o bien a continuar ejecutando la app, pero en modo offline, es decir, sin conexión a Internet. Si el usuario decide proseguir con la ejecución de la app en modo offline, se abrirá una base de datos interna y local al terminal y a la propia app, de manera que cuando el terminal tenga conexión a Internet los registros de esta base de datos local se transferirán automáticamente a la base de datos en la nube.

Por último, la función *onDeviceReady* programará una invocación periódica mediante la función *setInterval* a la función *comprobar_ubicacion_y_bluetooth_habilitados*, que como su propio nombre indica comprobará si el acceso a la ubicación o localización del terminal está permitido, así como si su interfaz Bluetooth está habilitada (y evidentemente actuará en consecuencia). Concretamente, si la interfaz Bluetooth está habilitada, si el terminal está emparejado al menos a un dispositivo Bluetooth externo (sea del tipo que sea, no necesariamente a uno con interfaz OBD2) y si el acceso a la ubicación o localización del terminal está permitido, esta función continuará la ejecución de la app, pero en caso contrario, es decir, en caso de que no se cumpla solamente uno de los tres anteriormente mencionados supuestos, la app mostrará un mensaje de error o advertencia que variará según cual haya sido la causa que haya impedido que la app continue ejecutándose con normalidad. Este mensaje de error instará al usuario a cerrar la app, a cambiar la configuración del terminal que está impidiendo que la app pueda ejecutarse con normalidad y a volverla a abrir.

En caso de que la configuración del terminal relativa a la interfaz Bluetooth y a su ubicación o localización sea correcta, como se ha comentado anteriormente, la app continuará ejecutándose con normalidad. Se invocará a la función *bluetoothSerial.list*, que creará una lista de todos los dispositivos Bluetooth externos con los que esté emparejado el terminal y la pondrá en la tabla de configuración para que el usuario pueda elegir uno de ellos. Además, también obtendrá de forma continua la ubicación o localización del terminal mediante invocaciones a determinadas funciones del plugin *Geolocation*.

Una vez que la función *onDeviceReady* ha hecho todo lo que se acaba de explicar, se puede decir que la app queda a la espera de que el usuario elija un dispositivo Bluetooth de la lista de dispositivos Bluetooth con los que está emparejado, de que introduzca los parámetros de configuración de la comunicación con la base de datos en la nube y de que pulse el botón *VALIDAR*.

Si el dispositivo Bluetooth seleccionado es un dispositivo Bluetooth con interfaz OBD2 e integrado ELM327 conectado a la interfaz OBD2 del vehículo y si los parámetros de configuración de la comunicación con la base de datos en la nube son correctos, la función *comprobar_validacion_o_detencion_configuracion* detectará que el usuario ha pulsado el botón validar y hará lo siguiente:

En primer lugar, asignará a las correspondientes variables del entorno de programación Javascript los valores de los parámetros introducidos por el usuario en la tabla de configuración (y la dirección MAC del dispositivo Bluetooth seleccionado) al mismo tiempo que guardará los valores de estos parámetros en el almacenamiento interno y no volátil reservado para la app.

En segundo lugar, invocará a la función *bluetoothSerial.connect*, que intentará conectarse con el dispositivo Bluetooth seleccionado.

En tercer lugar, programará una invocación periodica a la función

ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles cada tres segundos, que como su propio nombre indica comprobará cada vez que se la invoque si el acceso a la ubicación o localización del terminal está disponible, si la comunicación por Bluetooth con el dispositivo conectado a la interfaz OBD2 del vehículo no se ha interrumpido y si sigue existiendo conectividad a Internet.

En cuarto lugar, desactivará todos los elementos de la tabla de configuración excepto el botón *DETENER* (el botón *VALIDAR* también quedará desactivado) con el objetivo de que el usuario no pueda cambiar los valores de los parámetros de configuración del funcionamiento de la app una vez que estos ya han sido validados.

Por último, programará una invocación a la función *comunicacion_Internet_interrumpida* con un retraso igual al doble del periodo de almacenamiento de datos en la base de datos en la nube mediante la función *setTimeout* para que en caso de que la comunicación con la base de datos en la nube quede interrumpida el led verde parpadeante que indica que la comunicación con la base de datos en la nube se está llevando a cabo correctamente cambie de verde parpadeante a rojo fijo.

Por otra parte, la función *bluetoothSerial.connect*, en caso de que se haya podido establecer correctamente una conexión con el dispositivo Bluetooth conectado a la interfaz OBD2 del vehículo, indicará en la tabla correspondiente a los parámetros obtenidos del terminal que se ha establecido correctamente una conexión con el dispositivo Bluetooth con interfaz OBD2 conectado al vehículo. Además, programará una invocación con un retraso de cinco segundos a la función *comunicacion_bluetooth_interrumpida*, que se llevará a cabo en caso de que las funciones involucradas en la comunicación con el dispositivo Bluetooth conectado al vehículo no la cancelen a tiempo. También se invocará a la función *bluetoothSerial.subscribeRawData*, que a su vez invocará a la función *recibir_bluetoothExito* cada vez que el terminal reciba datos por la interfaz Bluetooth. Por último, programará una invocación periodica a la función *gestionar_comunicacion_bluetooth*, que se encargará, como su propio nombre indica, de gestionar la comunicación por Bluetooth con el dispositivo Bluetooth OBD2 conectado al vehículo (básicamente, esperar a recibir respuesta del dispositivo Bluetooth conectado al vehículo antes de enviarle el siguiente comando o petición de parámetro PID).

En lo que respecta a la función *ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles* mencionada anteriormente (cuya invocación se ha programado cada tres segundos), esta distinguirá dos casos principales: por un lado, si la ubicación del terminal está disponible (es decir, si la app sabe cuál es la ubicación del terminal), si la comunicación por Bluetooth con el adaptador OBD2 conectado al vehículo se está llevando a cabo y si existe una conexión válida a Internet, se actuará de una determinada manera; en cambio, si la ubicación del terminal está disponible y la comunicación por Bluetooth con el adaptador OBD2 conectado al vehículo se está llevando a cabo pero no existe una conexión válida a Internet en el terminal, entonces se actuará de otra manera.

Si existe conexión a Internet y ya existía de antemano una invocación periodica programada a la función *agregar_fila_bbdd_local* (para el caso en que se pase de no disponer de conexión a Internet a disponer de conectividad a la red), entonces se borra esta invocación programada periodicamente a la función *agregar_fila_bbdd_local*, dado que como ahora sí se dispone de conexión a Internet tiene más sentido almacenar los valores de los parámetros obtenidos del vehículo y del terminal en la base de datos en la nube en vez de en la base de datos local.

Por otra parte, siguiendo con el caso de que exista conexión a Internet, si no existía de antemano una invocación programada a la función *agregar_fila_bbdd_en_la_nube*, entonces se programa una invocación periodica a esta función con un periodo igual al valor introducido por el usuario en la tabla de configuración.

Por último, si la base de datos local no está vacía (caso en que se haya pasado de operar en modo offline a operar en modo online), entonces se invoca a la función *pasar_contenido_bbdd_local_a_bbdd_en_la_nubeExito*, que se encargará de recuperar uno por uno todos los registros que haya almacenados en la base de datos local e irlos enviando a la base de datos en la nube (además de vaciar la base de datos local una vez que termine de hacer esto último).

En caso contrario, es decir, en caso de que no exista conexión a Internet pero tanto la ubicación del terminal como la comunicación por Bluetooth estén disponibles, entonces, en caso de que la base de datos local no esté abierta, se invocará a la función *abrir_bbdd_local*, que se encargará de abrirla e inicializarla. Análogamente al caso anterior en el que sí existía conexión a Internet, si ya existía de antemano una invocación periodica programada a la función *agregar_fila_bbdd_en_la_nube* (caso en que se pase de operar en modo online a operar en modo offline), entonces esta se limpiará. Por último, si no existía una invocación periodica

programada a la función *agregar_fila_bbdd_local*, entonces se programará una invocación periodica a esta función con un periodo igual al valor introducido por el usuario en la tabla de configuración.

Además, cada vez que se la invoque, la función *ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles* invocará a su vez a la función *comprueba_conexion_a_Internet*, que como se ha comentado anteriormente determinará si el terminal tiene disponible una conexión válida a Internet.

3.1.2.-Estructura de directorios y ficheros.

El entorno de desarrollo Intel XDK crea por defecto, para cada nuevo proyecto, un directorio con el nombre del proyecto, dentro del cual estarán contenidos todos los archivos que este necesite para poder compilarse correctamente. El nombre que se le ha dado al proyecto, para simplificar, es *PFC* (Proyecto Fin de Carrera), de manera que el directorio del proyecto se llamará también de la misma manera. Dentro de él, caben destacar, entre el resto de directorios, el directorio *plugins* y el directorio *www*. El directorio *plugins* contendrá dentro la estructura de ficheros y directorios de cada uno de los plugins que se han añadido al proyecto, mientras que el directorio *www* contendrá los archivos del código fuente de la app (archivos HTML, Javascript y CSS) así como las imágenes usadas en la app.

Los plugins, por último, son un recurso software desarrollado para poder usar la API del sistema operativo móvil en cuestión desde la propia app. Son por tanto una manera de facilitar al programador de apps multiplataforma el acceso a los recursos hardware del sistema operativo móvil del terminal. Un esquema gráfico de la ubicación de los plugins en la arquitectura de una app multiplataforma desarrollada con Cordova puede observarse en la figura 4.

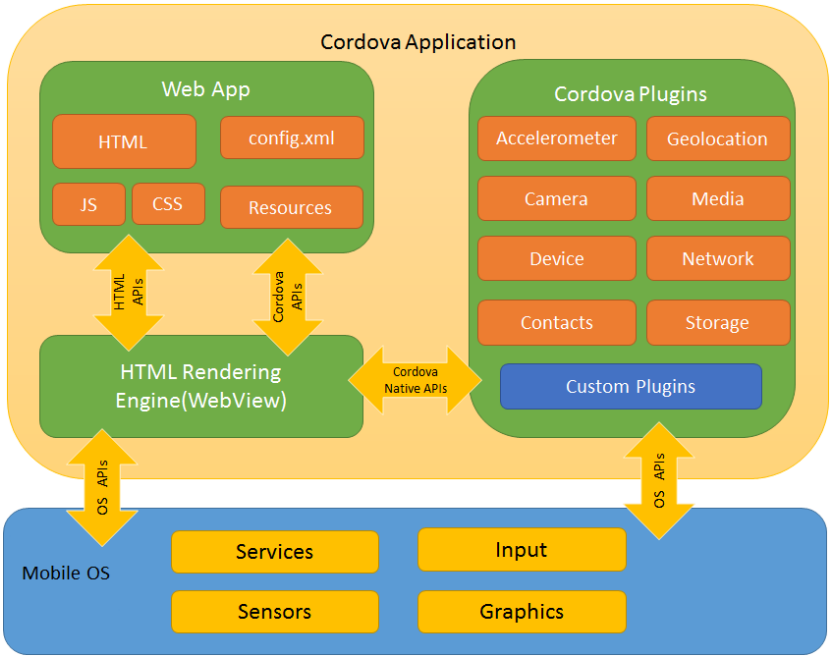

Figura 4: arquitectura de una app multiplataforma desarrollada con Cordova

3.1.3.-Estructura del código fuente de la app.

El código de una app desarrollada con Cordova, como se ha comentado anteriormente, son archivos escritos en los lenguajes HTML, CSS y Javascript. Esta app tendrá dos archivos HTML, un archivo CSS y un archivo Javascript (todos ellos ubicados dentro del directorio *www* como se ha comentado anteriormente).

Los archivos HTML son el archivo *main.html* y el archivo *index.html*. El archivo *index.html* es solamente un archivo HTML que redirecciona automáticamente al archivo *main.html*, por lo que el código HTML de la app se encontrará exclusivamente en el archivo *main.html*. Esto se ha hecho así para prevenir un error en la carga de la estructura de la web de la app, que en algunas ocasiones se daba e impedía iniciar la app correctamente. Por último, los estilos usados en el archivo *main.html* se encuentran en el archivo CSS *index.css*.

En cuanto al archivo con código javascript, este será el archivo *script.js*, que se incluirá en el archivo *main.html* usando la etiqueta <*script*>. La estructura del código Javascript de este archivo no es complicada. Por una parte, existe, nada más al comienzo del código, la declaración de una serie de variables de alcance global. Después de esta declaración de variables, se llama a la función *addEventListener*, que a su vez invocará a la función *onDeviceReady* una vez que la app esté lista y haya terminado de cargarse. Salvo por tres funciones que se comentarán a continuación y la declaración de variables que se ha hecho inicialmente, todo el código Javascript de la app se encuentra dentro de la función *onDeviceReady*, que es por tanto la función principal de la app.

Por último, además de la función *onDeviceReady*, que como se ha dicho contiene prácticamente todo el código de la app, y la declaración inicial de variables, entre la llamada a la función *addEventListener* y la función *onDeviceReady* existen tres funciones más que completan todo el código de la app. Estas funciones son las funciónes *cerrarApp*, *validar_configuracion* y *detener_configuracion*.

La función *cerrarApp* se ejecutará cuando se pulse el botón *SALIR*, situado en la parte más baja de la app. La función *validar_configuracion* se ejecutará cuando se pulse el botón *VALIDAR*, que validará la configuración relativa a la conexión con la base de datos en la nube y a la conexión con el vehículo de forma inalámbrica a través del adaptador Bluetooth con interfaz OBD2. Por último, la función *detener_configuracion* se ejecutará cuando se pulse sobre el botón *DETENER*, que detendrá la conexión con la base de datos en la nube, así como la conexión con el vehículo a través del adaptador Bluetooth con interfaz OBD2, al mismo tiempo que permitirá al usuario de la app cambiar los parámetros de configuración relativos a la conexión con la base de datos en la nube y los relativos a la conexión con el vehículo a través del adaptador Bluetooth con interfaz OBD2.

Dentro de la función *onDeviceReady*, que como se ha comentado anteriormente contiene prácticamente todo el código Javascript de la app, se encuentran todo el resto de funciones que ha sido necesario programar. Además, al inicio de la función *onDeviceReady*, se encuentra también todo el resto de declaraciones de variables de las que luego se hace uso. Por último, al final de esta función, se encuentran toda una serie de sentencias y de llamadas a funciones que provocan el inicio de la ejecución de la app.

3.2.-Funcionalidades o características de la app.

Se detallan y explican a continuación en los sucesivos sub-apartados las diferentes funcionalidades o características que se han programado en la app. Cuando se estime necesario y oportuno para complementar la explicación, se incluirán fragmentos de código fuente Javascript que pudieran ayudar a entender mejor o más claramente la funcionalidad o característica que se está describiendo. Además, al finalizar la explicación del funcionamiento de cada característica o funcionalidad, se incluirá al menos un diagrama explicativo del funcionamiento de la misma con el objetivo de facilitar la comprensión de esta por parte del lector.

3.2.1.-Comunicación por Bluetooth con el vehículo.

La comunicación por Bluetooth con el vehículo se ha basado, en primer lugar, en la imprementación de la función del código fuente Javascript encargada de listar los dispositivos Bluetooth con los que el terminal estuviera emparejado. Si la lista contiene al menos un elemento, la ejecución del programa sigue su curso normal, pero si no es así, se muestra un mensaje por pantalla que informa debidamente al usuario de que la app debe cerrarse.

En caso de que la lista contenga al menos un elemento (el dispositivo Bluetooth conectado a la interfaz OBD2 del vehículo), este se mostrará en la tabla de configuración para que el usuario pueda seleccionarlo. Si solamente hay un elemento en la lista de dispositivos Bluetooth emparejados con el terminal este estará seleccionado de antemano de forma automática.

Una vez que el usuario haya validado el resto de los parámetros de configuración junto con la selección del dispositivo Bluetooth conectado a la interfaz OBD2 del vehículo con el que quiere comunicarse, se invocará a una función cuyo cometido es conectarse con el adaptador Bluetooth conectado a la interfaz OBD2 del vehículo (función *bluetoothSerial.connect*). En esta función se hacen básicamente dos cosas. Por una parte, se subscribe a la recepción de datos por la interfaz Bluetooth, es decir, que cada vez que el terminal reciba datos por la interfaz Bluetooth se llamará a una determinada función (*recibir_bluetoothExito*) que se encargará de procesar y gestionar adecuadamente la información recibida. Por otra parte, se programa una llamada periodica a la función *gestionar_comunicacion_bluetooth*, que se encargará de gestionar el envío y la

recepción de información a través de la interfaz Bluetooth del terminal:

```
function gestionar_comunicacion_bluetooth()
    {
        if(esperando_recepcion_por_bluetooth == false)
        {
            enviar_comando_por_bluetooth();
            esperando_recepcion_por_bluetooth = true;
        }
    }
```

La gestión del envío y recepción de información a través de la interfaz Bluetooth se hace mediante el uso de una variable bandera (*esperando_recepcion_por_bluetooth*) que impide que se envíe nueva información a través de la interfaz Bluetooth del terminal antes de que se haya recibido respuesta de la última información o comando que se envio.

En caso de que sea posible enviar nueva información a través de la interfaz Bluetooth se invocará a la función *enviar_comando_por_bluetooth*, que se encargará de determinar qué comando o petición de PID corresponde enviarle al vehículo:

```
  function enviar_comando_por_bluetooth()
    {
        if(numero_de_comando == 1)
        {
            bluetoothSerial.write(comando_ATD, enviar_bluetoothExito);
        }
        if(numero_de_comando == 2)
        {
            bluetoothSerial.write(comando_ATZ, enviar_bluetoothExito);
        }
        if(numero_de_comando == 3)
        {
            bluetoothSerial.write(comando_AT_SP_0, enviar_bluetoothExito);
        }
        if(numero_de_comando == 4)
        {
            bluetoothSerial.write(comando_AT_E0, enviar_bluetoothExito);
        }
        if(numero_de_comando == 5)
        {
            bluetoothSerial.write(peticion_PID_VIN, enviar_bluetoothExito);
        }
        if(numero_de_comando == 6)
        {
            bluetoothSerial.write(peticion_PID_numero_de_errores, enviar_bluetoothExito);
        }
        if(numero_de_comando == 7)
        {
            bluetoothSerial.write(peticion_PID_codigos_de_error, enviar_bluetoothExito);
        }
        //A partir de aquí, las peticiones de PID que se repetirán indefinidamente:
        if(numero_de_comando == 8)
        {
            bluetoothSerial.write(peticion_PID_RPM, enviar_bluetoothExito);
        }
        if(numero_de_comando == 9)
        {
            bluetoothSerial.write(peticion_PID_temp_motor, enviar_bluetoothExito);
```

```
        }
        if(numero_de_comando == 10)
        {
            bluetoothSerial.write(peticion_PID_velocidad, enviar_bluetoothExito);
        }
        if(numero_de_comando == 11)
        {
            bluetoothSerial.write(peticion_PID_flujo_masa_aire, enviar_bluetoothExito);
        }
        if(numero_de_comando == 12)
        {
            bluetoothSerial.write(peticion_PID_posicion_acelerador, enviar_bluetoothExito);
        }
        if(numero_de_comando == 13)
        {
            bluetoothSerial.write(peticion_PID_temperatura_ambiente, enviar_bluetoothExito);
        }
        //Preparamos el envío del siguiente comando de la lista.
        numero_de_comando++;
        if(numero_de_comando > 13)
        {
            numero_de_comando = 8;
        }
    }
```

Para las primeras 7 invocaciones de la función *enviar_comando_por_bluetooth* se enviarán comandos de inicialización y también algunas peticiones de PID que solamente necesitan solicitarse una vez (como por ejemplo el número de identificación del vehículo, que como es lógico no cambiará durante el transcurso de la ejecución de la app). A partir de la invocación número 7, es decir, invocaciones octava y siguientes, las información enviada al vehículo a través de la interfaz Bluetooth del terminal se irá repitiendo periódicamente con la intención de registrar en la app los cambios que vayan experimentando los parámetros de funcionamiento del motor del vehículo (revoluciones por minuto, flujo de masa de aire, posición del acelerador, etc.).

Por último, cuando se reciba información a través de la interfaz Bluetooth, como se ha comentado anteriormente, se invocará a la función *recibir_bluetoothExito*. Su definición es muy larga, de manera que se incluye solamente en el Anexo A. Esta función hace básicamente dos cosas: por una parte, hace parpadear el led verde que indica que la comunicación por Bluetooth se está llevando a cabo; por otra parte, analiza la información recibida, determina a qué petición de parámetro PID corresponde, hace la conversión y los cálculos que sean precisos (dado que la información se recibe en código binario además de en una determinada codificación que depende del PID solicitado) y, por último, escribe los valores de los parámetros recibidos, ya en formato decimal, en la tabla correspondiente a los parámetros obtenidos de la interfaz OBD2 del vehículo.

Para una mayor claridad en la comprensión de todo lo explicado en este apartado, en la figura 5 se incluye un diagrama explicativo de la característica o funcionalidad de comunicación por Bluetooth de la app:

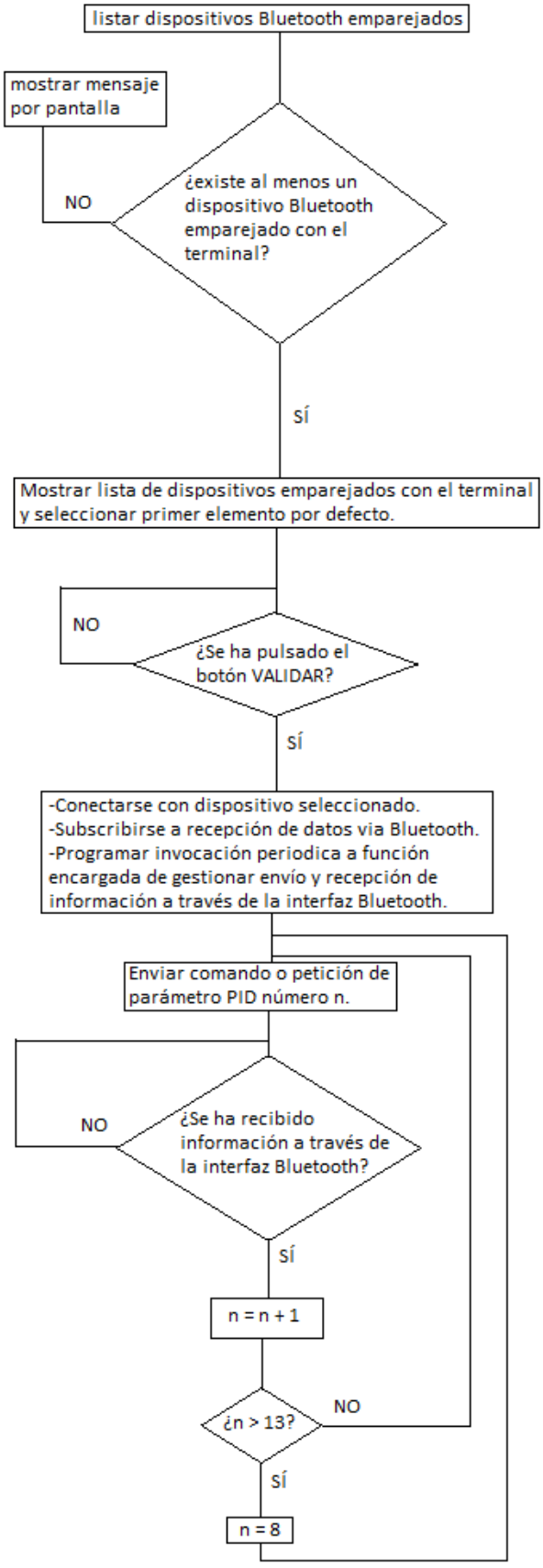

Figura 5: diagrama explicativo de la funcionalidad o característica de comunicación por Bluetooth de la app.

3.2.2.-Comunicación con la base de datos en la nube.

La comunicación con la base de datos en la nube se basa básicamente en la función *agregar_fila_bbdd_en_la_nube*. Como su definición es larga, solo se incluye en el Anexo A. En caso de que el terminal disponga de una conexión válida a Internet, se programaría una invocación periodica a esta función con un periodo igual al intervalo de tiempo medido en milisegundos introducido por el usuario en la tabla de configuración. Cada vez que se la invoca, esta función construye una petición HTTP POST con los valores que los parámetros tienen en el momento de su invocación. Finalmente, una vez construida, la peticion HTTP POST es enviada usando los parámetros de configuración de la base de datos en la nube introducidos por el usuario en la tabla de configuración.

Para una mejor comprensión de lo explicado en este apartado, en la figura 6 se incluye un diagrama de funcionamiento de esta característica o funcionalidad de la app:

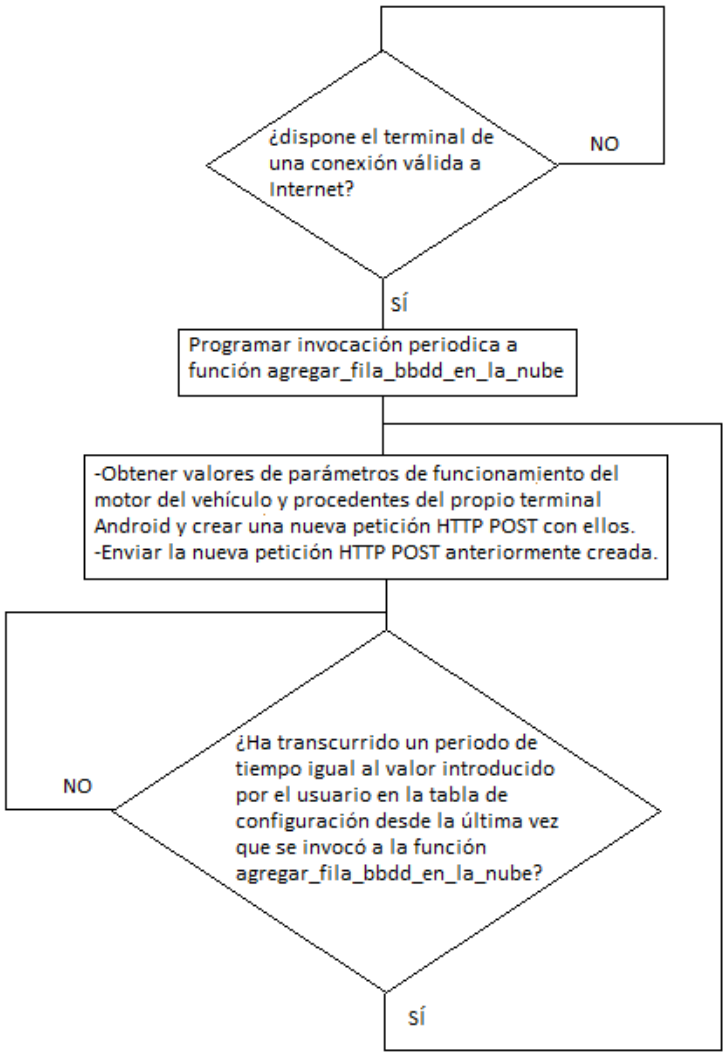

Figura 6: diagrama de funcionamiento de la comunicación de la app con la base de datos en la nube.

3.2.3.-Almacenamiento de datos en la base de datos local.

En caso de que el terminal no disponga de conexión a Internet, bien por un determinado periodo de tiempo o desde un primer momento, en vez de enviar los valores de los parámetros a la base de datos en la nube, estos se almacenarían en la base de datos local. El funcionamiento del almacenamiento en la base de datos local es análogo a la comunicación con la base de datos en la nube: en caso de que el terminal no disponga de conexión a Internet, se programaría una invocación periodica a esta función con un periodo igual al intervalo de tiempo

medido en milisegundos introducido por el usuario en la tabla de configuración. Cada vez que se la invoca, esta función contruye una instrucción SQL INSERT con los valores que los parámetros tienen en el momento de su invocación. Finalmente, una vez construida, la instrucción SQL INSERT es ejecutada en la base de datos local previamente creada, abierta y configurada.

Para una mejor comprensión de lo explicado en este apartado, en la figura 7 se incluye un diagrama de funcionamiento de esta característica o funcionalidad de la app.

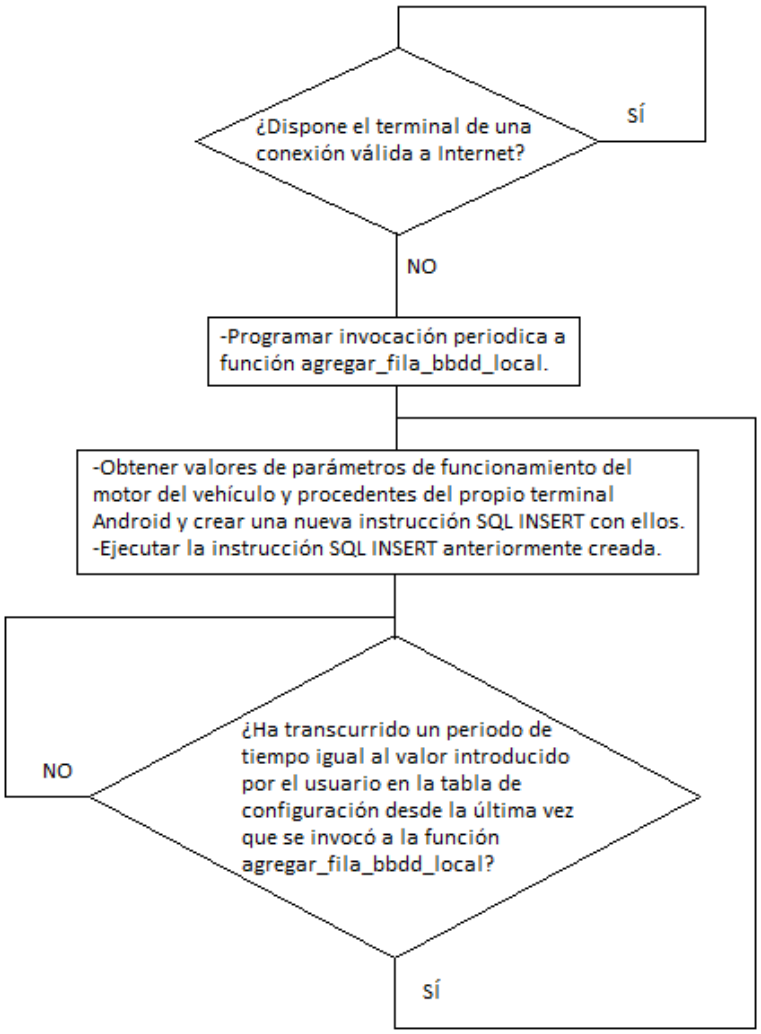

Figura 7: diagrama de funcionamiento del almacenamiento de datos en la base de datos local de la app.

3.2.4.-Paso de información de la base de datos local a la base de datos en la nube.

En caso de que el terminal pase de estar conectado a Internet a no estarlo, se dejarían de enviar peticiones HTTP POST a la base de datos en la nube y se comenzarían a ejecutar instrucciones SQL INSERT en la base de datos local. En caso de que ocurra lo contrario, es decir, en caso de que se pase de no disponer de conexión a Internet a sí estar conectado a la red, los parámetros almacenados en la base de datos local se traspasarían a la base de datos en la nube mediantes tantas nuevas peticiones HTTP POST como hicieran falta. Esta tarea la llevaría a cabo la función *pasar_contenido_bbdd_local_a_bbdd_en_la_nubeExito*, que recuperaría los datos de las instrucciones SQL INSERT anteriormente ejecutadas e iría construyendo tantas nuevas peticiones HTTP POST con ellos como fueran necesarias. La definición de esta función es demasiado larga, por lo que no se incluirá aquí, pero al igual que en los casos precedentes puede consultarse en el Anexo A.

Para una mayor y mejor comprensión de lo explicado en este apartado, se incluye en la figura 8 un diagrama de funcionamiento de esta característica o funcionalidad de la app:

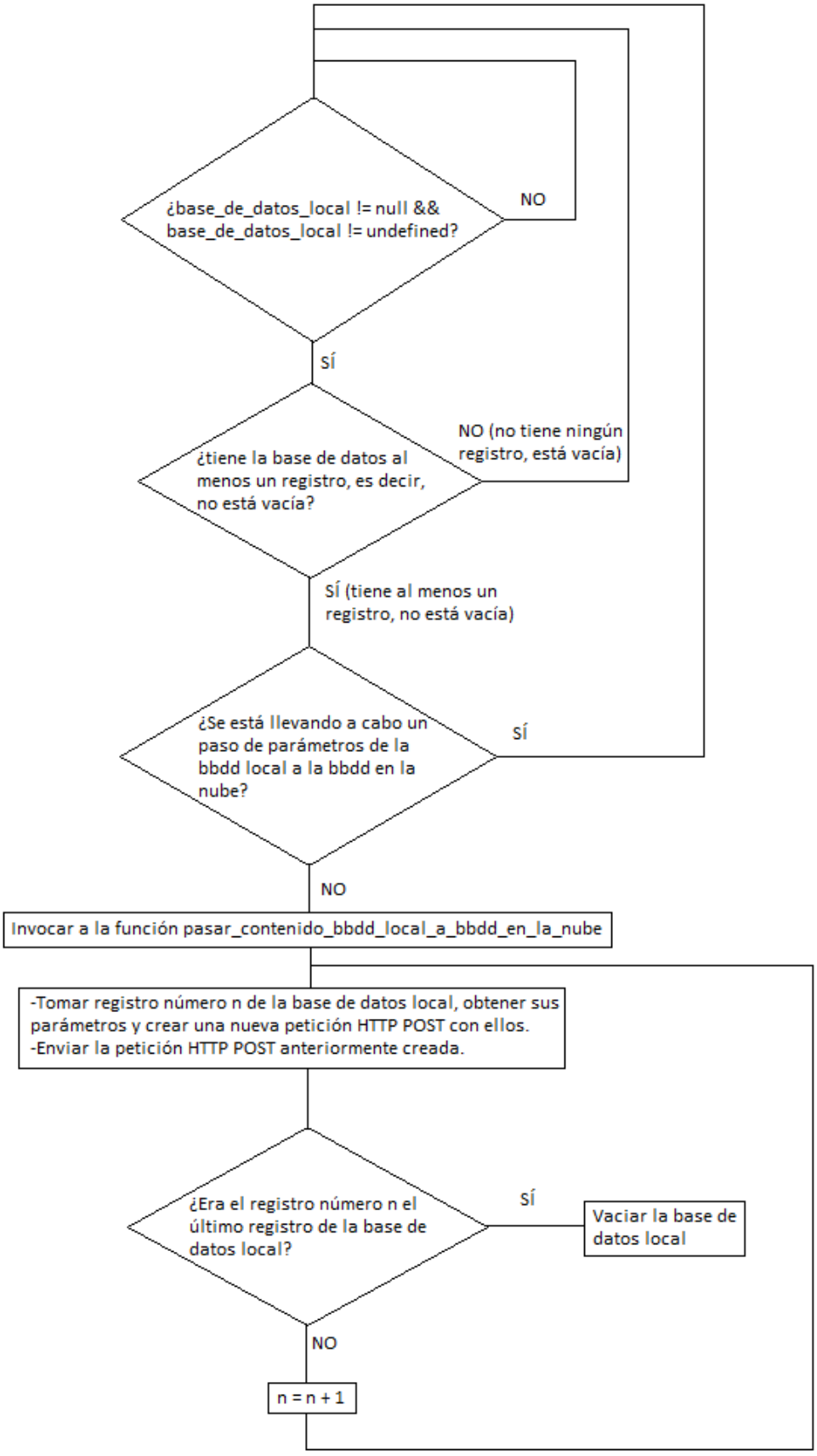

Figura 8: diagrama de funcionamiento del paso de información de la base de datos local a la base de datos en la nube.

3.2.5.-Obtención y actualización de la ubicación o localización del terminal.

Para la obtención y la actualización de la ubicación o localización del terminal se utilizará básicamente la función *obtener_ubicacionExito*. Por una parte, para la obtención de la ubicación, se invocará a la función *navigator.geolocation.getCurrentPosition*:

```
navigator.geolocation.getCurrentPosition(obtener_ubicacionExito, obtener_ubicacionError, {enableHighAccuracy: true});
```

En caso de que se produjera un error durante el proceso de la obtención del a ubicación o localización del terminal, se invocaría a la función *obtener_ubicacionError*, que guardaría en un registro el error generado. En cambio, si la obtención de la ubicación o localización del terminal se ha podido llevar a cabo sin errores (que de hecho será lo más probable y habitual), se invocaría a la función *obtener_ubicacionExito*:

```
function obtener_ubicacionExito(position)

    {

        //Guardamos los parámetros o las coordenadas de la ubicación del terminal en las variables globales correspondientes:

        latitud = position.coords.latitude;

        longitud = position.coords.longitude;

        altitud = position.coords.altitude;

        precision_latitud = position.coords.accuracy;

        precision_longitud = position.coords.accuracy;

        precision_altitud = position.coords.altitudeAccuracy;

        //Multiplicamos por ratio_conversion_velocidad (3,6) para obtener un valor expresado en km/h en vez de en m/s.

        velocidad = position.coords.speed*ratio_conversion_velocidad;

        //Escribimos las coordenadas o los parámetros de la ubicación del terminal en la correspondiente tabla del documento HTML:

        document.getElementById("latitud").innerHTML = latitud;

        document.getElementById("longitud").innerHTML = longitud;

        document.getElementById("altitud").innerHTML = altitud;

        document.getElementById("precision_latitud").innerHTML = precision_latitud;

        document.getElementById("precision_longitud").innerHTML = precision_longitud;

        document.getElementById("precision_altitud").innerHTML = precision_altitud;

        document.getElementById("velocidad").innerHTML = velocidad;

        //Cambiamos el icono y el texto del estado de la obtención de la ubicación del terminal:

        document.getElementById("estado_obtencion_ubicacion").innerHTML = "<p align='center'><img src='img/icono_GPS_estatico.png'> <br>
Ubicaci&oacute;n obtenida con &eacute;xito.</p>";

        //Si es la primera vez que se obtiene la ubicación se indica que la ubicación ya sí está disponible:

        if(ubicacion_disponible == false)

        {

            ubicacion_disponible = true;

        }

    }
```

Esta función toma en primer lugar los valores de los parámetros relativos a la ubicación o localización del terminal y los almacena en variables globales accesibles desde cualquier punto de la app. A continuación, estos valores se escriben en la tabla correspondiente a los parámetros obtenidos del terminal Android. Por último, en caso de que sea la primera vez que se obtiene la ubicación o localización del terminal, se cambia de valor *false* a valor *true* una variable bandera que indica si la ubicación o localización del terminal Android está disponible y que será de utilidad en otros puntos del código Javascript de la app.

Por otra parte, para la actualización de la ubicación o localización del terminal, se invocará a la función *navigator.geolocation.watchPosition*:

```
watchLocationId = navigator.geolocation.watchPosition(actualizar_ubicacionExito, actualizar_ubicacionError, {enableHighAccuracy: true});
```

En caso de que el terminal cambie de ubicación o localización y se genere un error al intentar obtenerla, se invocaría a la función *actualizar_ubicacionError*, que a su vez invocaría a la función *obtener_ubicacionError* anteriormente mencionada:

```
function actualizar_ubicacionError(error)

    {

        obtener_ubicacionError(error);

    }
```

Por el contrario, en caso de que el terminal cambie de ubicación o localización y esta se pueda obtener (que de hecho será lo más probable y habitual), se invocaría a la función *actualizar_ubicacionExito*, que a su vez invocará a la función *obtener_ubicacionExito* anteriormente mencionada:

```
function actualizar_ubicacionExito(position)
    {
        obtener_ubicacionExito(position);
    }
```

Para una mayor y mejor comprensión de lo explicado en este apartado, se incluye a continuación en la figura 9 un diagrama explicativo de esta funcionalidad o característica de la app.

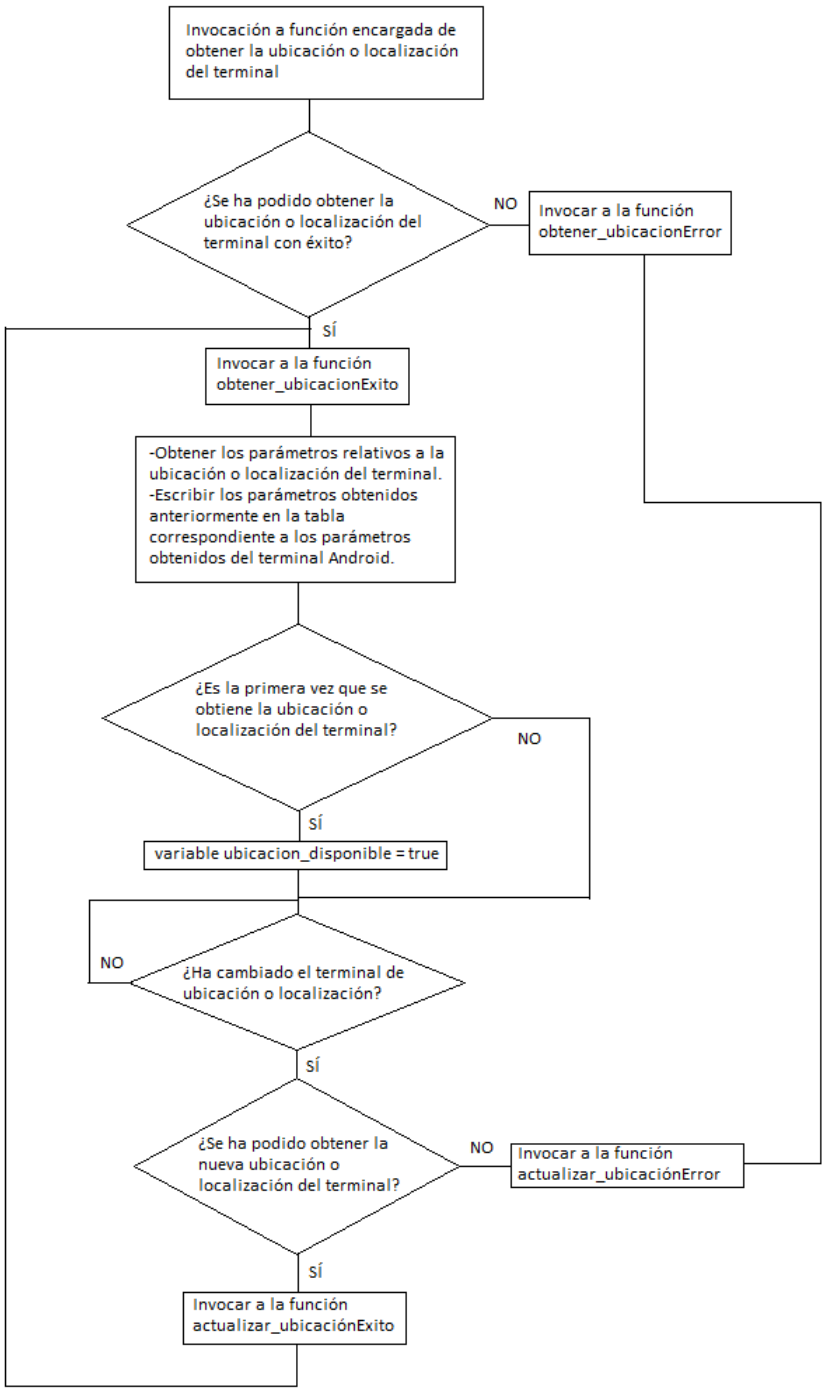

Figura 9: diagrama de funcionamiento de la obtención y actualización de la localización o ubicación del terminal.

3.2.6.-Funcionamiento de los botones *VALIDAR*, *DETENER* y *SALIR*.

Para explicar el funcionamiento de los botones *VALIDAR*, *DETENER* y *SALIR* distinguiremos por una parte los botones *VALIDAR* Y *DETENER* y, por otra, el botón *SALIR*.

En primer lugar, el funcionamiento del botón *SALIR* es el más sencillo de todos, dado que solamente hay una función implicada en su funcionamiento, la función *cerrarApp* y, además, esta función también es muy simple.

El código HTML del botón *SALIR* se ha programado de manera que se invoca a la función *cerrarApp* cuando se pulsa en él:

```
<button onclick="cerrarApp()" id="gordo">SALIR</button>
```

Por su parte, la función cerrarApp, además de encontrarse en el interior de la función *onDeviceReady* (que por cierto es la función en cuyo interior se encuentran la mayoría de las funciones del código fuente Javascript de la app), también se encuentra fuera de ella, lo que permite invocarla desde cualquier punto del código fuente Javascript de la app. Es posible invocarla incluso desde un ámbito ajeno al código fuente Javascript, como por ejemplo desde un archivo de código fuente HTML, tal y como es el caso.

Se incluye a continuación la definición de la función cerrarApp, cuyo único cometido, como su propio nombre indica, es cerrar la app, distinguiendo además entre varios casos en caso de que los hubiera:

```
function cerrarApp()
{
    if (navigator.app)
    {
        navigator.app.exitApp();
    }
    else if (navigator.device)
    {
        navigator.device.exitApp();
    }
    else
    {
        window.close();
    }
}
```

En segundo lugar, están los botones *VALIDAR* y *DETENER*, cuyo funcionamiento es más complejo que el del botón *SALIR*. Cuando se pulsa alguno de estos botones, se invoca a la función *validar_configuracion* (caso del botón *VALIDAR*) o a la función *detener_configuracion* (caso del botón *DETENER*). La definición de estas funciones es también muy simple como en el anterior caso del botón *SALIR*. Lo que ocurre es que la complejidad del funcionamiento de los botones *VALIDAR* y *DETENER* no acaba ahí, tal y como se detallará a continuación.

Cuando se pulsa el botón *VALIDAR*, tal y como se ha comentado, se invocará a la función *validar_configuracion*, que desactivará el botón *VALIDAR* y activará el botón *DETENER* con el objetivo de que no se pueda volver a pulsar el botón *VALIDAR* hasta que no se haya pulsado el botón *DETENER*. Además, se le asignará el valor *true* a la variable bandera *configuracion_validada* con el objetivo de que el resto del código fuente Javascript de la app pueda tener constancia de que se ha pulsado el botón *VALIDAR*, y por último se le asignarán a la variable global *datos_configuracion* los valores de los parámetros de configuración introducidos por el usuario en la tabla de configuración que previamente han sido recibidos como parámetro en la variable local *datos*:

```
function validar_configuracion(datos)
{
    //Cambiamos el botón "VALIDAR" a deshabilitado y el botón "DETENER" a habilitado (parámetro disabled):
    document.getElementsByName("boton_validar")[0].disabled = true;
    document.getElementsByName("boton_detener")[0].disabled = false;
    //Hacemos que los parámetros de configuración estén disponibles globalmente en cualquier punto del código de la app:
    datos_configuracion = datos;
```

```
    //Mandamos una señal a la función comprobar_validacion_o_detencion_configuracion:
    configuracion_validada = true;
}
```

Hay que comentar que con el objetivo tanto de detectar si se ha invocado a la función *validar_configuración* como con el objetivo de detectar si se ha invocado a la función *detener_configuracion* se ha programado previamente una invocación periodica temporizada a la función *comprobar_validacion_o_detencion_configuracion*. Esta función será invocada cada medio segundo y comprobará los valores de las variables globales *configuracion_validada* y *configuracion_detenida*. En caso de que detecte que el valor de la variable *configuracion_validada* es *true* este hecho indicaría que se ha invocado a la función *validar_configuracion*, y que por tanto se ha pulsado el botón *VALIDAR*. Si esto ocurre, la ejecución de la función *comprobar_validacion_o_detencion_configuracion* entraría dentro de la sentencia *if* que comprueba si el valor de la variable *configuracion_validada* es *true* y llevaría a cabo las siguientes acciones:

En primer lugar, guardaría en el almacenamiento local reservado para la app dentro de la memoria del terminal los valores de los parámetros introducidos por el usuario en la tabla de configuración con el objetivo de que estos sean recordados en las siguientes y sucesivas ejecuciones de la app para evitarle así al usuario tener que volver a introducirlos.

En segundo lugar, invocaría a la función *bluetoothSerial.connect*, que como su propio nombre indica iniciaría el procedimiento de conexión con el dispositivo Bluetooth conectado a la interfaz OBD2 del vehículo.

En tercer lugar, la función *comprobar_validacion_o_detencion_configuracion* desactivaría todos los elementos del formulario de la tabla de configuración excepto el botón *DETENER*, que quedaría activado para que el usuario pueda pulsarlo en caso de que desee detener la comunicación de la app tanto con el dispositivo Bluetooth conectado a la interfaz OBD2 del vehículo como con la base de datos en la nube.

Y en cuarto y último lugar, se cambiaría el valor de la variable *configuracion_validada* a *false* con el objetivo de que todo lo que anteriormente se ha llevado a cabo solamente se haga una única vez hasta que se vuelva a pulsar sobre el botón *VALIDAR*, dado que como se ha explicado anteriormente la función *comprobar_validacion_o_detencion_configuracion* será invocada cada medio segundo, con lo que si el valor de la variable *configuracion_validada* no ha cambiado de *true* a *false* de una ejecución o invocación a la siguiente todo lo que se ha llevado a cabo en el interior de la sentencia *if* que comprueba si el valor de la variable *configuracion_validada* es *true* se volvería a ejecutar inexorablemente.

Se incluye a continuación, para una mejor comprensión de lo anteriormente explicado, la sentencia *if* de la función *comprobar_validacion_o_detencion_configuracion* que comprueba si el valor de la variable *configuracion_validada* es *true*:

```
if(configuracion_validada == true)
        {
            configuracion_interrumpida = false;
            //Guardamos en las variables de configuración los valores de los parámetros de configuración correspondientes introducidos en
la tabla de configuración.
            periodo_almacenamiento_bbdd = datos_configuracion.periodo_almacenamiento_bbdd.value;
            //Guardamos en el almacenamiento local el periodo de almacenamiento en la base de datos para que se recuerde en la siguiente
ejecución de la app:
            window.localStorage.setItem('periodo_almacenamiento_bbdd', periodo_almacenamiento_bbdd);
            dispositivo_bluetooth_seleccionado = datos_configuracion.dispositivo_bluetooth.value;
            email_cuenta_Zoho = datos_configuracion.email_cuenta_Zoho.value;
            //Guardamos en el almacenamiento local el email de la cuenta Zoho para que se recuerde en la siguiente ejecución de la app:
            window.localStorage.setItem('email_cuenta_Zoho', email_cuenta_Zoho);
            authtoken = datos_configuracion.authtoken.value;
            //Guardamos en el almacenamiento local el token de autenticación para que se recuerde en la siguiente ejecución de la app:
            window.localStorage.setItem('authtoken', authtoken);
            nombre_bbdd = datos_configuracion.nombre_bbdd.value;
            //Guardamos en el almacenamiento local el nombre de la base de datos para que se recuerde en la siguiente ejecución de la
app:
            window.localStorage.setItem('nombre_bbdd', nombre_bbdd);
            nombre_tabla_bbdd = datos_configuracion.nombre_tabla_bbdd.value;
```

```
            //Guardamos en el almacenamiento local el nombre de la tabla de la base de datos para que se recuerde en la siguiente
ejecución de la app:

            window.localStorage.setItem('nombre_tabla_bbdd', nombre_tabla_bbdd);

            MAC_adaptador_bluetooth = dispositivos_bluetooth_emparejados[dispositivo_bluetooth_seleccionado].address;

            //alert("Parametros configuracion:\n" + periodo_almacenamiento_bbdd + "\n" + dispositivo_bluetooth_seleccionado + "\n" +
email_cuenta_Zoho + "\n" + authtoken + "\n" + nombre_bbdd + "\n" + nombre_tabla_bbdd + "\n" + MAC_adaptador_bluetooth);

            //Nos conectamos con la ECU (Engine Control Unit) del vehículo a través del adaptador Bluetooth seleccionado:

            bluetoothSerial.connect(MAC_adaptador_bluetooth, conexion_bluetoothExito, conexion_bluetoothError);

            //Comprobamos una vez cada tres segundos si tanto la ubicación como la comunicación por Bluetooth y la conexión a Internet
están disponibles para empezar a agregar datos a la base de datos almacenada localmente o en la nube:

            ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles_id =
setInterval(ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles, tres_segundos);

            //Hacemos que los valores de la tabla de configuración ya no se puedan cambiar (parámetro disabled = true):

            formulario_configuracion = document.getElementsByName("formulario_configuracion")[0];

            //alert("Número de elementos del formulario de configuracion: " + formulario_configuracion.elements.length);

            for(i=0; i<formulario_configuracion.elements.length; i++)

            {

                //Deshabilitamos todos los elementos del formulario salvo el número 7: el botón "DETENER", que debe quedar habilitado:

                if(i != numero_elemento_boton_detener)

                {

                   formulario_configuracion.elements[i].disabled = true;

                }

                else if(i == numero_elemento_boton_detener)

                {

                    formulario_configuracion.elements[i].disabled = false;

                }

            }

            //Si existe conexión a Internet al pulsar el boton "VALIDAR", programamos la invocación de la función
comunicacion_Internet_interrumpida justo cuando transcurran dos veces el periodo de almacenamiento de datos en la base de datos en la
nube, tiempo suficiente para que se cree al menos una nueva petición POST que limpie la invocación de esta función antes de que esta se
lleve a cabo:

            if(conexion_Internet_disponible == true)

            {

                comunicacion_Internet_interrumpida_id = setTimeout(comunicacion_Internet_interrumpida, periodo_almacenamiento_bbdd*2);

            }

            //Cambiamos el valor de la bandera para que todo lo que se tiene que ejecutar al validar la configuración solamente se
ejecute una única vez:

            configuracion_validada = false;

        }
```

En caso de que la *función comprobar_validacion_o_detencion_configuracion* detecte que se ha pulsado el botón *DETENER*, su ejecución entraría dentro de la sentencia *if* que comprueba si el valor de la variable *configuracion_detenida* es *true*, en cuyo caso llevaría a cabo lo siguiente:

En primer lugar, invocaría a la función *bluetoothSerial.disconnect*, que como su propio nombre indica iniciaría el procedimiento de desconexión del dispositivo Bluetooth conectado a la interfaz OBD2 del vehículo.

En segundo lugar, si exitieran invocaciones periodicas programadas a las funciones *agregar_fila_bbdd_local* o *agregar_fila_bbdd_en_la_nube*, las limpiaría.

En tercer lugar, activaría todos los elementos del formulario de la tabla de configuración excepto el botón *DETENER*, que se dejaría desactivado.

En cuarto lugar, programaría una invocación con un retraso de 2 segundos a la función *borrar_valores_parametros_obtenidos_interfaz_OBD2_vehiculo* con la intención de que trancurridos 2 segundos desde que se detectó que se pulsó el botón *DETENER* se borren los valores de los parámetros obtenidos de la interfaz OBD2 del vehículo, dado que al haber pulsado el botón *DETENER* y haberse detenido la comunicación con el vehículo los valores de estos parámetros no estarían actualizados.

Por último, en caso de que el terminal disponga de conectividad a Internet, se invocaría a la función *comunicacion_Internet_interrumpida*, que indicaría con un led rojo fijo en la tabla relativa a los parámetros obtenidos del terminal que se ha interrumpido la comunicación con el servidor de la base de datos en la nube.

Se incluye a continuación, para una mejor comprensión de lo anteriormente explicado, la sentencia *if* de la función *comprobar_validacion_o_detencion_configuracion* que comprueba si el valor de la variable *configuracion_detenida* es *true*:

```
if(configuracion_detenida == true)

        {

            configuracion_interrumpida = true;

            //alert("Configuracion detenida");

            //Detenemos la invocación programada cada 3 segundos a la función que comprueba si está disponible la ubicación, la
comunicación por Bluetooth y la conexión a Internet:

            clearInterval(ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles_id);

            //Nos desconectamos del adaptador Bluetooth:

            bluetoothSerial.disconnect(desconexion_bluetoothExito, desconexion_bluetoothError);

            //En caso de que exista una invocación programada a la función agregar_fila_bbdd_local la limpiamos:

            if(invocacion_programada_a_funcion_agregar_fila_bbdd_local == true)

            {

                clearInterval(agregar_fila_bbdd_local_id);

                invocacion_programada_a_funcion_agregar_fila_bbdd_local = false;

            }

            //En caso de que exista una invocación programada a la función agregar_fila_bbdd_en_la_nube la limpiamos:

            if(invocacion_programada_a_funcion_agregar_fila_bbdd_en_la_nube == true)

            {

                clearInterval(agregar_fila_bbdd_en_la_nube_id);

                invocacion_programada_a_funcion_agregar_fila_bbdd_en_la_nube = false;

            }

            //Hacemos que los parámetros de la tabla de configuración se puedan volver a editar:

            formulario_configuracion = document.getElementsByName("formulario_configuracion")[0];

            for(i=0; i<formulario_configuracion.elements.length; i++)

            {

                //Habilitamos todos los elementos del formulario salvo el número 7: el botón "DETENER", que debe quedar deshabilitado:

                if(i != numero_elemento_boton_detener)

                {

                    formulario_configuracion.elements[i].disabled = false;

                }

                else if(i == numero_elemento_boton_detener)

                {

                    formulario_configuracion.elements[i].disabled = true;

                }

            }

            //Borramos las filas de la columna "Valor" de la tabla de los parámetros obtenidos de la interfaz OBD2 del vehículo escepto
las filas correspondientes a los parámetros número de identificación del vehículo (VIN), número de errores detectados en el vehículo y
códigos de error de los errores detectados en el vehículo, ya que no tiene sentido mantener valores obsoletos de parámetros obtenidos de
la interfaz OBD2 del vehículo una vez que la comunicación por Bluetooth ha sido detenida.

            setTimeout(borrar_valores_parametros_obtenidos_interfaz_OBD2_vehiculo, time_out_detencion_configuracion);

            //Si al pulsar el botón "DETENER" existe una conexión a Internet Dado que se va a detener la comunicación con la base de
datos en la nube, se invoca a la función comunicacion_Internet_interrumpida para cambiar el led verde parpadeante a rojo y se limpia al
mismo tiempo la invocación temporizada a esta función que se hizo con anterioridad al pulsar el botón "VALIDAR":

            if(conexion_Internet_disponible == true)

            {

                comunicacion_Internet_interrumpida();

                clearTimeout(comunicacion_Internet_interrumpida_id);

            }

            //Cambiamos el valor de la bandera de entrada al cuerpo del if para que este solamente se ejecute una vez.

            configuracion_detenida = false;

        }
```

Por último, para una mayor y mejor comprensión de todo lo explicado en este apartado, se incluyen a continuación tres diagramas: uno del funcionamiento del botón *SALIR*, otro del funcionamiento del botón *VALIDAR* y otro del funcionamiento del botón *DETENER*.

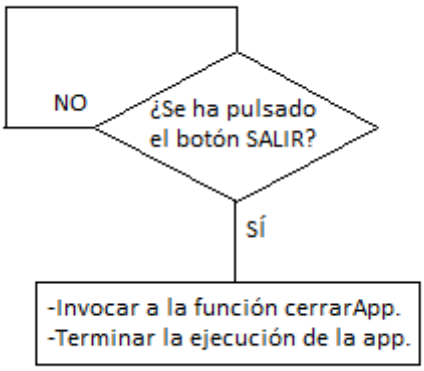

Figura 10: diagrama de funcionamiento del botón *SALIR*.

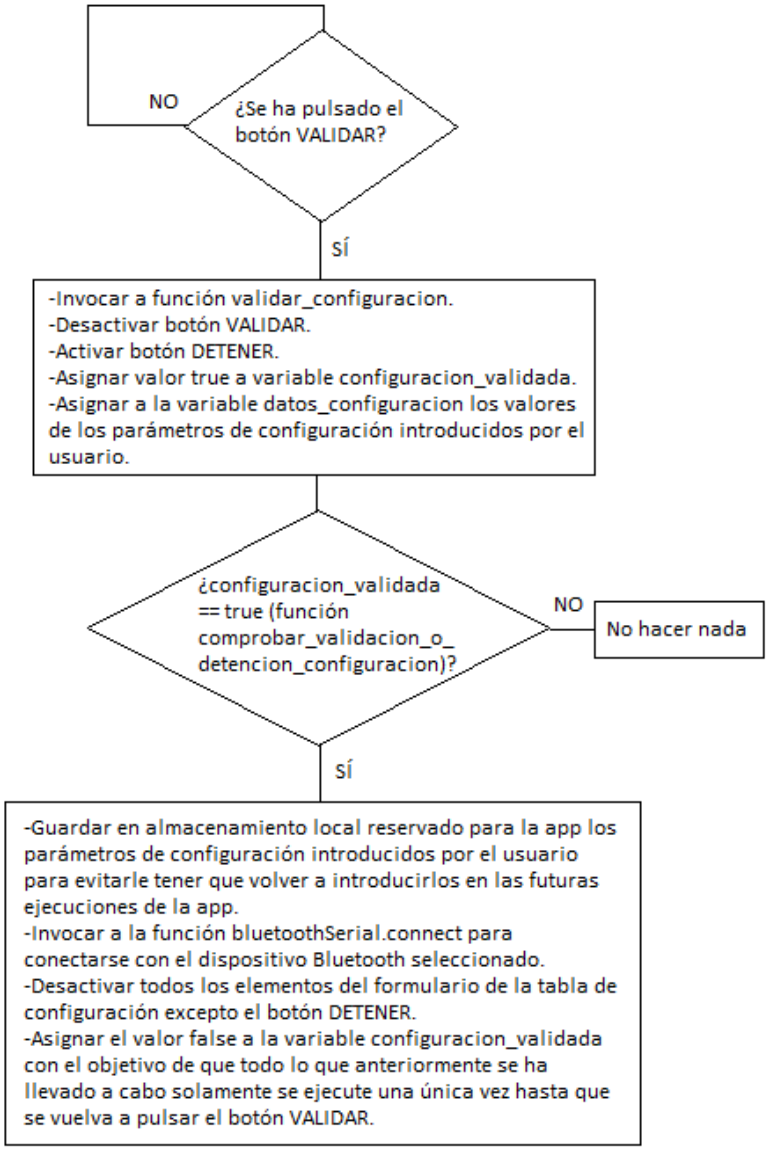

Figura 11: diagrama de funcionamiento del botón *VALIDAR*.

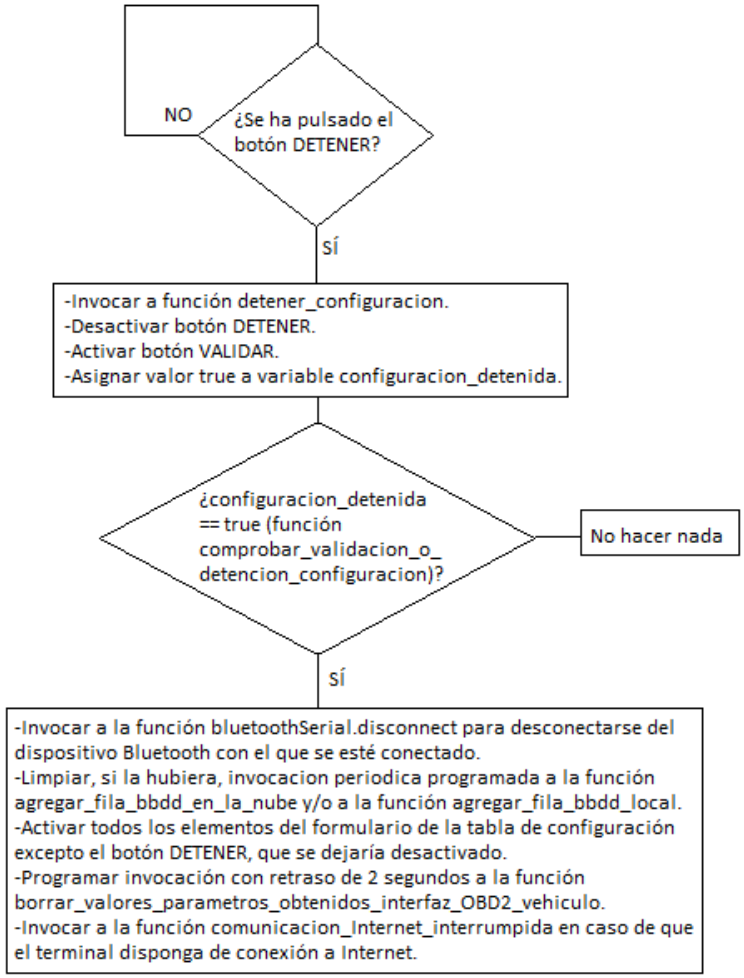

Figura 12: diagrama de funcionamiento del botón DETENER.

3.3.-Problemas encontrados.

Se describen a continuación por riguroso orden de ocurrencia los problemas que se han ido originando durante el desarrollo del proyecto.

3.3.1.-Incoherencia entre la versión simulada y la versión en tiempo de ejecución de la app.

El primer problema de significativa relevancia que se encontró fue que la versión en tiempo de ejecución y la simulada no se comportaban de la misma manera. Cuando decimos versión en tiempo de ejecución se hace referencia a la versión de la app en formato apk instalada en el terminal Android, mientras que cuando se hace mención a la versión simulada se está haciendo referencia a la misma versión de la app simulada en el propio entorno de desarrollo Intel XDK.

Se podría resumir la diferencia de comportamiento entre la versión simulada de la app y la versión en tiempo de ejecución en que había llamadas a funciones que sí se hacían durante la simulación pero que por el motivo que se expondrá a continuación no se llegaban a materializar en tiempo de ejecución.

La razón de ser de este comportamiento erróneo radicaba en la forma en la que creó inicialmente la estructura del proyecto de forma automática por el entorno de desarrollo Intel XDK: este había generado una serie de archivos-plantilla HTML, Javascript y CSS que estaban destinados a encajar en otro tipo de proyecto de programación de app multiplataforma, como por ejemplo un proyecto de app de IoT (Internet of Things o Internet de las cosas).

La solución que se adoptó fue crear un nuevo proyecto desde cero con una plantilla que fuera lo más general y simple posible (o incluso sin ningún tipo de plantilla en absoluto) e ir copiando y pegando el contenido de los archivos del proyecto anterior, el proyecto que ya tenía de antemano algunas cosas predefinidas de forma

predeterminada.

3.3.2.-Falta de la opción de comunicación por Bluetooth de la versión para Linux del simulador *OBDSim*.

El software simulador *OBDSim* es el que se usó para llevar a cabo la mayoría de las pruebas de comunicación por Bluetooth de la app. Este simulador por software, como su propio nombre deja entrever, simula o emula a nivel software el comportamiento de la *ECU* (Engine Control Unit o unidad de control del motor) de un vehículo real con interfaz OBD2. Para ser más precisos, este simulador simula o emula por software el comportamiento de un integrado ELM327, que es el integrado que usan los dispositivos de conversión de interfaz OBD2 a interfaz Bluetooth (ver figura 13).

Figura 13: dispositivo de conversión de interfaz OBD2 a interfaz Bluetooth. Dispone de un integrado ELM327 en su interior.

El motivo por el que se ha usado un simulador de la *ECU* (Engine Control Unit o unidad de control del motor) de un vehículo es el de no tener que usar un vehículo real con interfaz OBD2 cada vez que se quisiera hacer alguna prueba de la comunicación por Bluetooth entre la app y un vehículo equipado con un dispositivo igual o similar al mostrado en la figura 13. En su lugar, por razones prácticas y de eficiencia, se usó el simulador por software *OBDSim* junto con un adaptador Bluetooth con interfaz USB conectado a un ordenador personal que enviaba al dispositivo con el que estaba emparejado la información que recibía del simulador *OBDSim*.

Sin embargo, no todo fue "coser y cantar", ya que en un principio la versión para Linux de este software (que fue la que en un principio se pensó usar por motivos de compatibilidad) no tenía disponible la funcionalidad que permitía la comunicación por Bluetooth con el otro extremo, es decir, con el terminal Android emparejado al ordenador personal a través del adaptador Bluetooth con interfaz USB.

La versión que en un principio se compiló en una distribución Ubuntu disponía de las siguientes opciones (comando "obdsim --help"):

```
Usage: obdsim [params]
   [-g|--generator=<name of generator>
       [-s|--seed=<generator-seed>]
       [-d|--customdelay=<ecu delay(ms)>]
     ]
   [-q|--logfile=<logfilename to write to>]
   [-V|--elm-version=<pretend to be this on ATZ>]
   [-D|--elm-device=<pretend to be this on AT@1>]
   [-L|--list-protocols]
   [-p|--protocol=<OBDII protocol>]
   [-o|--launch-logger]
   [-c|--launch-screen] ["EXIT" or C-a,k to exit]
   [-t|--tty-device=<real /dev/ entry to open>]
   [-e|--genhelp=<name of generator>]
   [-l|--list-generators]
   [-n|--benchmark=<seconds>]
   [-v|--version] [-h|--help]
```

Pero según la herramienta *man*, las opciones disponibles del simulador *OBDSim* son en realidad las siguientes (comando “man obdsim”):

```
OPTIONS
       -g|--generator <generator-name>
              Choose  a     generator.  A list of valid ones is output by --help.
              See section titled MULTIPLE ECUS below for more information.

       -s|--seed <seed>
              Generator-specific seed. See section titled PLUGIN  SEEDS     below
              for  more information. The -s option must immediately follow the
              generator

       -d|--customdelay <delay-in-ms>
              Generator-specific delay. This is effectively a processing  time
              for  the     ECU  it is being added for. The -d option must immedi-
              ately follow the generator

       -l|--list-generators
              Print a terse list of compiled in generators

       -L|--list-protocols
              Print a list of all protocols

       -p|--protocol <OBDII protocol>
              Launch as this protocol. Protocol is of form  [A]{digit},     where
              optional     "A"  prefix  means  automatic  and  the          digit is from
              --list-protocols

       -n|--benchmark <time>
              Change time to print samplerate to stdout.  'samples'  are  suc-
              cessful  value  returns, not AT commands or NO DATA/? responses.
              'queries' are any and all client queries. Argument  is  in  sec-
              onds, zero to disable.

       -q|--logfile <logfile>
              Write all serial comms to this logfile

       -o|--launch-logger
              Takes  an     [admittedly weak and hard-coded] attempt at launching
              obdgpslogger attached to the simulator in question. POSIX     only.

       -c|--launch-screen
              Takes  an     [admittedly weak and hard-coded] attempt at launching
              screen attached to the simulatorin  question.  To  close  that
              screen window, use ctrl-a, k. POSIX only.

       -t|--tty-device
              Instead  of  opening  a  pty,  try  to  open  this entry in /dev
              instead. POSIX only.

       -w|--com-port <comport>
              Specify virtual com port to be used on windows [eg "COM1"]. Win-
              dows only.

       -e|--genhelp <generator-name>
              Print out help for the specified generator, and exit.

       -V|--elm-version <version string>
              Pretend to be this when someone resets with ATZ or similar

       -D|--elm-device <device string>
              Pretend to be this when someone calls AT@1

       -b|--bluetooth
              Listen on bluetooth. See section titled BLUETOOTH below

       -v|--version
              Print out version number and exit.

       -h|--help
              Print out help and exit.
```

Como puede apreciarse, la opción de comunicación por Bluetooth marcada en color azul está presente en la información proporcionada por la herramienta *man*, pero no en las opciones disponibles en la propia herramienta *OBDSim* (comando “obdsim --help”), por lo que a nivel práctico no se podrá utilizar la funcionalidad de comunicación por Bluetooth de este simulador.

Indagando e investigando un poco, se llega a la hipótesis de que la falta de la opción de comunicación por Bluetooth del software puede ser debida a la no inclusión de ciertas librerías necesarias en tiempo de compilación. Y efectivamente, mirando el código fuente de la herramienta, vemos que hay una serie de directivas de preprocesado que determinan si compilar incluyendo la opción de comunicación por Bluetooth en el resultado final o no:

```
#ifdef HAVE_BLUETOOTH
#include "bluetoothsimport.h"
#endif //HAVE_BLUETOOTH
...
#ifdef HAVE_BLUETOOTH
                "   [-b|--bluetooth]\n"
#endif //HAVE_BLUETOOTH
```

Además de estas directivas de preprocesado, también están implicadas en la inclusión de la opción de comunicación por Bluetooth otras instrucciones de compilación relativas a la herramienta *cmake*:

```
SET(OBD_SIM_DISABLE_BLUEZ false CACHE BOOL "Disable bluetooth support in
obdsim")
IF(NOT OBD_SIM_DISABLE_BLUEZ)
        CHECK_SYMBOL_EXISTS(BTPROTO_RFCOMM
                bluetooth/bluetooth.h
                        HAVE_BLUETOOTH)
        IF(HAVE_BLUETOOTH)
                MESSAGE(STATUS "Enabling bluetooth obdsim port")
                ADD_DEFINITIONS(-DHAVE_BLUETOOTH)
        ENDIF(HAVE_BLUETOOTH)
ENDIF(NOT OBD_SIM_DISABLE_BLUEZ)
```

En definitiva, instalando en el sistema operativo Ubuntu el paquete "libbluetooth-dev" y volviendo a compilar el resultado ya es satisfactorio: la opción de comunicación por Bluetooth del simulador *OBDSim* ahora sí está disponible.

3.3.3.-Problema al intentar conectar por Bluetooth la app con el ordenador PC-compatible.

Se encontraron dificultades o problemas a la hora de establecer una conexión por Bluetooth entre el terminal Android y un ordenador PC-compatible. En un principio, dado que la versión del simulador *OBDSim* que se iba a usar era la versión correspondiente al sistema operativo Linux, la conexión se intentó establecer en este sistema.

El primer problema que se encontró fue que al llamar en el código Javascript a la función correspondiente relativa a la conexión por Bluetooth con otro dispositivo (en el caso que nos ocupa, un ordenador PC-compatible), esta función siempre llamaba a la función que correspondía a haber encontrado un error al intentar establecer una conexión entre ambos dispositivos, nunca a la función que correspondía a haber podido emparejar o conectar con éxito un dispositivo con el otro. Es decir, siempre se acababa llamando a la función *conexión_bluetoothError*, nunca a la función *conexión_bluetoothExito*:

```
bluetoothSerial.connect(MAC_adaptador_bluetooth, conexion_bluetoothExito, conexion_bluetoothError);
```

En un primer momento se pensó en que el adaptador Bluetooth estaba mal configurado en el Sistema operativo Linux, por lo que se ejecutaron en la línea de comandos de este sistema operativo un par de comandos de configuración de la interfaz Bluetooth que dieran información a este respecto:

```
juanma@Toshiba-Juanma:~$ hcitool dev
Devices:
hci0 00:11:22:98:76:54

juanma@Toshiba-Juanma:~$ hciconfig
hci0: Type: Primary  Bus: USB
BD Address: 00:11:22:98:76:54  ACL MTU: 1021:4  SCO MTU: 180:1
UP RUNNING
RX bytes:521 acl:0 sco:0 events:26 errors:0
TX bytes:611 acl:0 sco:0 commands:27 errors:2
```

Pero de la información proporcionada por ambos commandos de configuración se se concluye que el adaptador Bluetooth está correctamente configurado en Linux.

Se piensa en un principio por tanto en descartar de alguna otra manera que el adaptador Bluetooth no pueda funcionar correctamente en el sistema operativo, así como que el error, de alguna u otra manera, radique en la

propia app. Para lo primero, se intenta hacer una transferencia de ficheros entre el terminal Android y el ordenador PC-compatible con sistema operativo Linux, mientras que para lo segundo se intenta hacer funcionar el simulador *OBDSim* con otra app de obtención de datos de *ECUs* (Engine Control Units) de vehículos extensamente probada: *Torque Lite*.

El resultado de la primera prueba es que no es posible llevar a cabo una transferencia de ficheros entre el terminal Android y el ordenador PC-compatible con sistema operativo Linux, mientras que el resultado de la segunda prueba es que tampoco es posible establecer comunicación entre la app *Torque Lite* ejecutándose en el terminal Android y el simulador *OBDSim* ejecutándose en el ordenador PC-compatible.

Concluyéndose por tanto que los problemas de configuración o comunicación entre el terminal Android y el ordenador PC-compatible con sistema operativo Linux son patentes, se intenta probar por último el adaptador Bluetooth en el mismo ordenador PC-compatible, solo que en el sistema operativo Windows en vez de en el sistema operativo Linux. Señalar que el adaptador Bluetooth con interfaz USB es "Plug&Play", es decir, que no necesita la instalación de ningún tipo de driver adicional ni la edición de ningún tipo de configuración específica para poder hacerlo funcionar correctamente, sino que solamente hay que conectarlo para poder empezar a usarlo. De hecho, no trae consigo ningún tipo de medio de almacenamiento externo de información en el que haya algún tipo de drivers o manual de usuario que indique cómo configurar el dispositivo.

El resultado de la instalación del adaptador Bluetooth con interfaz USB en el sistema operativo Windows del ordenador PC-compatible es muy llamativo: el dispositivo genera un conflicto de hardware en el propio sistema operativo. Para no perder más tiempo con este adaptador Bluetooth con interfaz USB, se concluye que el adaptador Bluetooth USB está defectuoso: es necesario adquirir uno nuevo, a ser posible de mejor calidad.

Una vez habiendo adquirido un nuevo adaptador Bluetooth con interfaz USB y de mejor calidad que el anterior, la primera prueba que se lleva a cabo es la última que se hizo con el anterior adaptador Bluetooth con interfaz USB: probarlo en el sistema operativo Windows. El resultado es que, a diferencia del anterior adaptador Bluetooth defectuoso con interfaz USB, este nuevo adaptador Bluetooth no solo no provoca ningún conflicto de hardware en el sistema operativo Windows, sino que además permite la transferencia de ficheros entre el terminal Android y el ordenador PC-compatible con sistema operativo Windows. Se concluye por tanto que, a diferencia del anterior adaptador Bluetooth con interfaz USB que se descartó, este nuevo adaptador Bluetooth con interfaz USB no está defectuoso, aunque esto último no necesariamente implique que vaya a poder funcionar tan fácilmente en Linux como lo ha hecho en Windows.

El siguiente paso sería por tanto intentar hacer funcionar este nuevo adaptador Bluetooth con interfaz USB en el sistema operativo Linux. Sin embargo, el resultado no es satisfactorio: se obtienen tantos errores como con el adaptador Bluetooth con interfaz USB defectuoso que se descartó.

Habida cuenta de que con este adaptador Bluetooth USB no se consigue establecer comunicación entre el terminal Android y el sistema operativo Linux del ordenador PC-compatible, se intenta buscar información a través de Internet sobre cómo hacer funcionar el simulador *OBDSim* en el sistema operativo Windows en vez de en el sistema operativo Linux, ya que en Windows este nuevo adaptador Bluetooth con interfaz USB parecía funcionar correctamente (incluso se pudieron transferir ficheros entre el terminal Android y el sistema operativo Windows del ordenador PC-compatible).

Buscando en Internet se encuentra el enlace [13], que explica muy clara y brevemente cómo hacer funcionar el simulador *OBDSim* en el sistema operativo Windows, que era lo que se estaba buscando.

Según lo que se explica en una de las respuestas del enlace anterior, en primer lugar, hay que emparejar el terminal Android con el sistema operativo Windows del ordenador PC-compatible. Y en segundo y último lugar, es necesario configurar el puerto COM de entrada de la interfaz Bluetooth del sistema operativo con un determinado nombre (por ejemplo, COM5).

Una vez hecho esto, tan solo hay que tener en cuenta que a la hora de ejecutar la versión para Windows del simulador *OBDSim* hay que hacerlo usando la opción -w seguida del puerto COM de entrada de la interfaz Bluetooth que se configuró anteriormente (por ejemplo, COM5):

```
C:\Program Files (x86)\OBDsim>obdsim.exe -w COM5
```

Una vez hecho esto y estando emparejado de antemano el terminal Android con el sistema operativo Windows del ordenador PC-compatible, se comprueba que ya *Torque Lite* puede efectivamente comunicarse con el simulador *OBDSim* ejecutándose en el sistema operativo Windows. Además, también se comprueba que la

función que se llama en el código Javascript de la app del proyecto ya no llama siempre a la función conexion_bluetoothError, sino que ahora llama siempre a la función *conexion_bluetoothExito*, lo que indica no solamente que el error de comunicación a través de la interfaz Bluetooth ha sido solucionado, sino que además este error no era causado por la app.

3.3.4.-Problema con la implementación de protocolos de comunicación con la *ECU* del vehículo.

A la hora de implementar en la app los protocolos de comunicación con la *ECU* del vehículo, se han intentado implementar en la app los protocolos de comunicación entre el integrado ELM327 del adaptador Bluetooth con interfaz OBD2 conectado al vehículo y la *ECU* del vehículo, por lo que el resultado, aunque afortunadamente no ha sido nefasto, sí que ha sido totalmente imprevisible. Se ha cometido un error de concepto a este respecto, ya que no se ha tenido en cuenta que precisamente el integrado ELM327 presente en el adaptador Bluetooth con interfaz OBD2 hace de interfaz (a través del protocolo de comunicación Bluetooth) entre el terminal Android (que usa un protocolo serie para comunicarse con el adaptador Bluetooth con interfaz OBD2) y la *ECU* del vehículo, que a su vez se comunica con el integrado ELM327 con otro tipo de protocolos y estándares de comunicación que nada tienen que ver con el protocolo serie que usa el terminal Android para comunicarse con el integrado ELM327. Viendo en este sentido que el proyecto no avanzaba, el tutor corrigió o explicó este error de concepto con ayuda de un esquema que dibujó a mano alzada cuya adaptación se muestra en la figura 14.

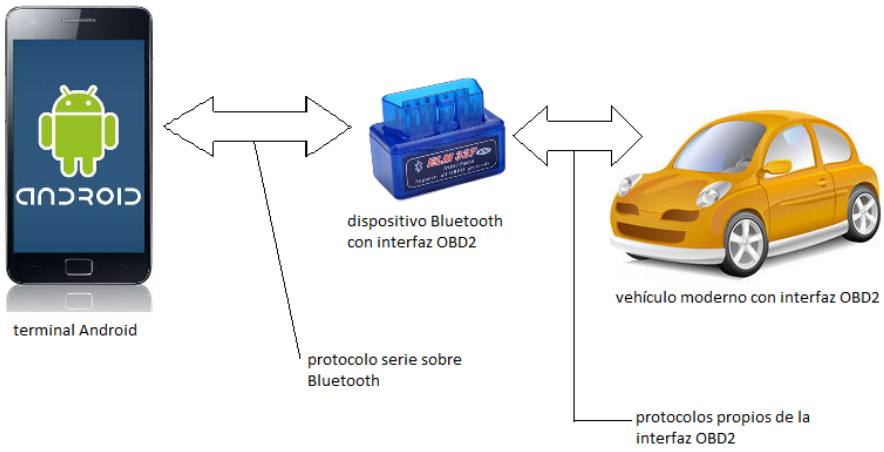

Figura 14: esquema de conexión o de comunicación entre la *ECU* del vehículo y el terminal Android a través del adaptador Bluetooth con interfaz OBD2 e integrado ELM327.

3.3.5.-Problema de comunicación con el simulador *OBDSim*.

Los comandos que se le enviaban al simulador *OBDSim* no eran bien entendidos por este, ya que no respondía adecuadamente a los mismos o no lo hacía según lo esperado. Este hecho se comprobó que sucedía de esta manera porque la respuesta del simulador a unos mismos comandos no era la misma si esos comandos se enviaban desde la app del proyecto que si se enviaban desde la app BlueTerm, app que dicho sea de paso emula un terminal serie implementado sobre la base del protocolo Bluetooth: algo así como una línea de comandos remota en el dispositivo con el que se está emparejado a través de Bluetooth.

Después de indagar y de hacer pruebas se llega a la conclusión de que a diferencia de lo que hace la app del proyecto, la app BlueTerm (Bluetooth Terminal) envía al final de cada comando un carácter de salto de línea y otro de retorno de carro, por lo que se llega a la conclusión de que el simulador necesita de la existencia de estos carácteres al final del envío de cada comando. Igualmente, al responder, también se comprueba que el simulador responde con otro par de carácteres de salto de línea y de retorno de carro, algo que, dicho sea de paso, vino bien saber de cara a la programación de la lectura de los comandos de respuesta de la *ECU* del vehículo (o en su defecto, del simulador).

3.3.6.-Error de comunicación con el vehículo real.

Se produjo error de comunicación a la hora de probar la app en un vehículo real tanto con la app del proyecto como con la app BlueTerm. El error consistía básicamente en la imposibilidad de establecer una comunicación efectiva entre cualquiera de las apps anteriormente mencionadas y el vehículo real. Aunque no se podía establecer comunicación, sin embargo, sí que se podía establecer conexión entre el vehículo real y la app del proyecto (no así entre el vehículo real y la app BlueTerm). El hecho de que tampoco se pudiera establecer comunicación (y ni siquiera conexión) entre la app BlueTerm y el vehículo real sugería en un principio que este nefasto error no estaba en la propia app del proyecto, lo que era motivo de alivio...

Se pensó en un principio en algún tipo de incompatibilidad entre el adaptador Bluetooth OBD2 que se estaba usando y el propio vehículo real. Sin embargo, esta hipotética causa u origen del error fue rápidamente descartada al comprobarse que la app *Torque Lite* podía comunicarse y sincronizarse perfectamente con el vehículo real usando el mismo adaptador Bluetooth OBD2 que en un principio estaba supuestamente impidiendo la comunicación entre la app del proyecto y el vehículo real.

Finalmente, despues de hacer muchas pruebas de depuración se llegó a la conclusión de que el motivo por el cual la app del proyecto no podía comunicarse coherentemente con un vehículo real estaba en que por la razón que sea, un vehículo real, a diferencia del simulador *OBDSim*, no responde a las peticiones de los parámetros de funcionamiento o de estado del vehículo que se le solicitan con un carácter de retorno de carro seguido de un carácter de nueva línea, sino que envía sin ningún tipo de separación la respuesta al parámetro que se le solicitó. Esto hacía que nunca se detectaran las respuestas enviadas por el vehículo real a los parámetros solicitados, ya que siempre se esperaban un carácter de nueva línea y un carácter de retorno de carro en las respuestas recibidas por parte del vehículo real, carácteres que nunca eran debidamente detectados por la app del proyecto sencillamente porque nunca se enviaban.

Por lo tanto, hubo que programar doblemente la gestión de las respuestas a los parámetros solicitados tanto al vehículo real como al simulador *OBDSim*. Por cada tipo de parámetro solicitado, se programó la detección de la respuesta enviada por el vehículo real (sin carácter de retorno de carro ni de nueva línea al comienzo de la misma) y la detección de la respuesta enviada por el simulador *OBDSim* (con un carácter de retorno de carro y otro de nueva línea al comienzo de la misma).

Una vez que se hizo esto el problema quedó solucionado y se pudo comprobar como la app del proyecto detectaba las respuestas a los parámetros solicitados enviadas tanto por el vehículo real como por el simulador *OBDSim*.

3.3.7.-Error de compilación al agregar nuevo plugin.

Llegados a un cierto punto en el desarrollo de la app fue necesario agregar un nuevo plugin, pero el resultado fue nefasto: el software Intel XDK ya no era capaz de compilar adecuadamente el código fuente de la app. Se investigó un poco al respecto buscando en la World Wide Web con el buscador Google y se visitaron varios foros que en principio podrían haber arrojado un poco de luz al respecto, aunque no fue así. En la figura 15 se puede observar una captura de pantalla del error que lanzaba *Intel XDK* cuando se intentaba construir (*build*) o compilar la app. En ella, en su parte izquierda, se observa como aparece en rojo el indicador "BUILD FAILED", que indica que el proceso de construcción o compilación de la app ha fallado.

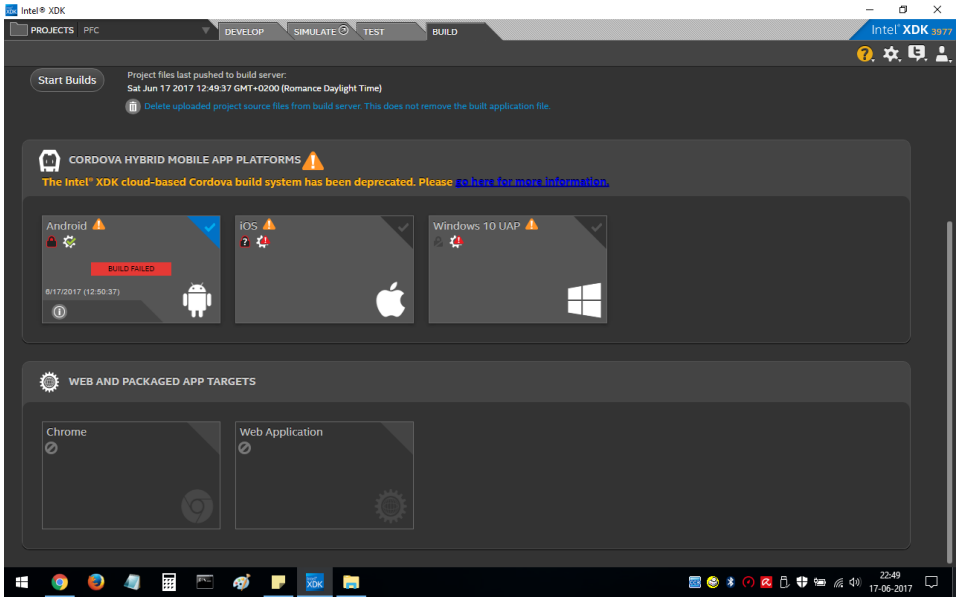

Figura 15: captura de pantalla del error de compilación surgido al añadir un nuevo plugin.

Este error era ajeno al código fuente de la app escrito por el alumno y por tanto podría decirse que no era competencia suya, aunque por otra parte era evidente que afectaba al desarrollo de la app, por lo que debía ser corregido. Algunos foros de Internet aludían a la existencia de una cierta incoherencia o conflicto existente en el archivo "AndroidManifest.xml", pero después de indagar al respecto se optó por ahorrar tiempo y esfuerzo simplemente desinstalando el plugin y volviendolo a instalar, algo que por cierto no solucionó el problema.

El plugin que recientemente se había agregado y que presuntamente estaba causando el error de compilación había sido el plugin "Diagnostic", así que se optó por desinstalarlo e intentar volver a compilar. El resultado: el error de compilación ya no se producía, con lo que estaba claro que su causante era el plugin "Diagnostic".

Sin embargo, el plugin "Diagnostic" no se había añadido o agregado por gusto, sino que era necesario usarlo con el objetivo de que la app pudiera determinar en tiempo de ejecución si la interfaz Bluetooth del terminal estaba habilitada y si se podía tener acceso a la información relativa a la ubicación o localización del terminal. Así que había que buscar otra alternativa en forma de plugin para poder tener acceso a esta información del hardware del terminal. Se encontraron varias alternativas que en principio podían sustituir al plugin "Diagnostic", pero finalmente se optó por probar una versión reducida de este mismo plugin que solamente obtenía la información del hardware del terminal relativa a la ubicación o localización de este. Para obtener la información relativa a la activación o desactivación de la interfaz Bluetooth del terminal se optó por usar la función "isEnabled" del plugin "Bluetooth Serial", que como su propio nombre indica proporciona información acerca de la activación o desactivación de la interfaz Bluetooth del terminal.

Una vez agregada esta versión reducida del plugin "Diagnostic" ya solo faltaba volver a intentar compilar o construir la app para determinar si el error persistía. Y el resultado fue satisfactorio: el error ya no tenía lugar, de manera que se podía continuar con el desarrollo de la app con la tranquilidad de que llegado a un cierto punto en el que fuera necesario compilarla ya no se volvería a producir este error que en un principio estaba impidiendo compilar o construir la versión ejecutable de la app a partir del código fuente programado.

4 PRUEBAS Y RESULTADOS

Ser independiente de la opinión pública es la primera condición formal para lograr algo grande.

- Friedrich Hegel -

En este capítulo se detallarán y explicarán minuciosamente las pruebas que se han llevado a cabo tanto con la propia app del proyecto como también con otras apps con capacidad para establecer comunicación con otro dispositivo emparejado a través de la interfaz Bluetooth del terminal Android. Se incluyen en los sucesivos apartados las pruebas realizadas tanto con la ayuda del simulador *OBDSim* (mencionado en capítulos anteriores) como con la ayuda de un vehículo real. En ambos casos se han hecho pruebas usando en el terminal Android la propia app del proyecto, la app *BlueTerm* (mencionada también en capículos anteriores) y la app *Torque Lite* (mencionada igualmente en capítulos anteriores).

4.1.-Preliminares.

Antes de empezar a explicar las diferentes pruebas que se han llevado a cabo es necesario explicar el contexto tecnológico (tanto hardware como software) en el que se han llevado a cabo cada una de las pruebas.

Por ejemplo, en lo que respecta al terminal Android, la versión de este sistema operativo móvil presente en el terminal fue la 4.2.2. El terminal utilizado fue un Alcatel OneTouch POP C7, terminal de gama media.

Por otra parte, el vehículo usado para llevar a cabo las pruebas en un vehículo real fue un Citroën Xsara Picasso del año 2002, que evidentemente disponía de interfaz o puerto OBD2.

A la hora de hacer las pruebas con la ayuda del simulador *OBDSim*, como se ha explicado en capítulos anteriores, se pensó en un principio en usar la versión para Linux de este software, pero finalmente, debido a problemas de incompatibilidad hardware, se optó por usar la versión para Windows. La versión para Windows del software *OBDSim* que se usó fue la 0.16, que emulaba o, mejor dicho, simulaba, la versión 1.3a del integrado ELM327. Además, el ordenador PC-compatible que se usó fue un Toshiba Satellite Pro con sistema operativo Windows 10, 4 GB de memoria RAM y procesador Intel Pentium Dual Core a 2 GHz.

Por último, en lo que respecta al dispositivo Bluetooth con interfaz OBD2 usado en la/s prueba/s llevadas a cabo con un vehículo real, la versión del integrado ELM327 presente en el aparato fue la 2.1.

4.2.-Pruebas llevadas a cabo con el simulador *OBDSim*.

Distinguimos en este apartado tres categorías o tipos de pruebas: la/s llevada/s a cabo usando la app *BlueTerm*, la/s llevada/s a término usando la app *Torque Lite* y por último la/s realizada/s usando la propia app del proyecto.

4.2.1.-Prueba de envío de comandos básicos con la app *BlueTerm*.

En esta prueba se trataba fundamentalmente de comprobar la comunicación por Bluetooth mediante protocolo serie entre el terminal Android y el ordenador PC-compatible en el que se ejecutaba el software simulador *OBDSim*. Mediante la app *BlueTerm* pudieron enviarse al simulador *OBDSim* comandos básicos de inicialización, así como solicitud de parámetros de funcionamiento del vehículo que el simulador *OBDSim* supo entender correctamente. El hecho de que el simulador *OBDSim* entendiera los comandos y las solicitudes de parámetros de funcionamiento del vehículo que se le enviaban era prueba de que todo funcionaba correctamente y según lo previsto, además de un aliciente para programar en la app del proyecto el envío de comandos de inicialización y de solicitudes de parámetros de funcionamiento del vehículo de la misma manera que se hacía en la app *BlueTerm*. La figura 16 es una captura de la pantalla del terminal Android usado en la

prueba, en la que se pueden observar los comandos de inicialización y las solicitudes de parámetros de funcionamiento del vehículo enviados desde la app *BlueTerm* al simulador *OBDSim*, así como la respuesta recibida de este software. En la figura 17, por otra parte, se observa la respuesta o la reacción del simulador *OBDSim* a la recepción de tales comandos de inicialización y solicitudes de parámetros de funcionamiento del vehículo.

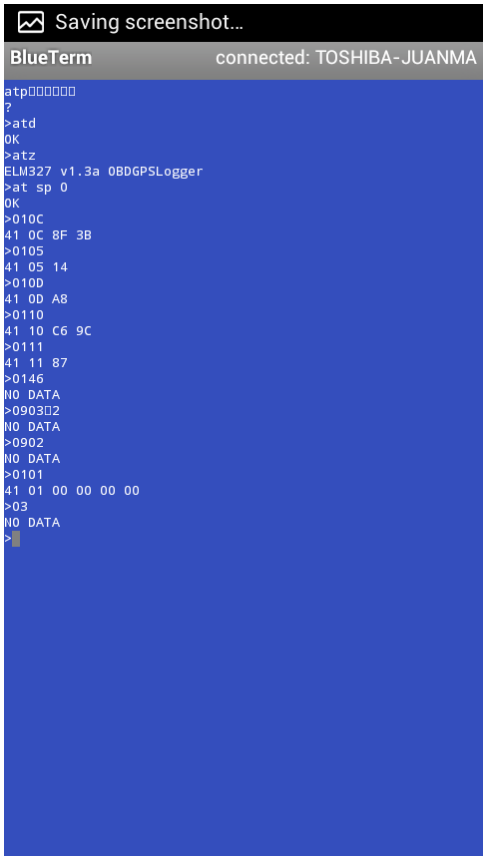

Figura 16: captura de pantalla del terminal Android usado en la prueba con la app *BlueTerm*.

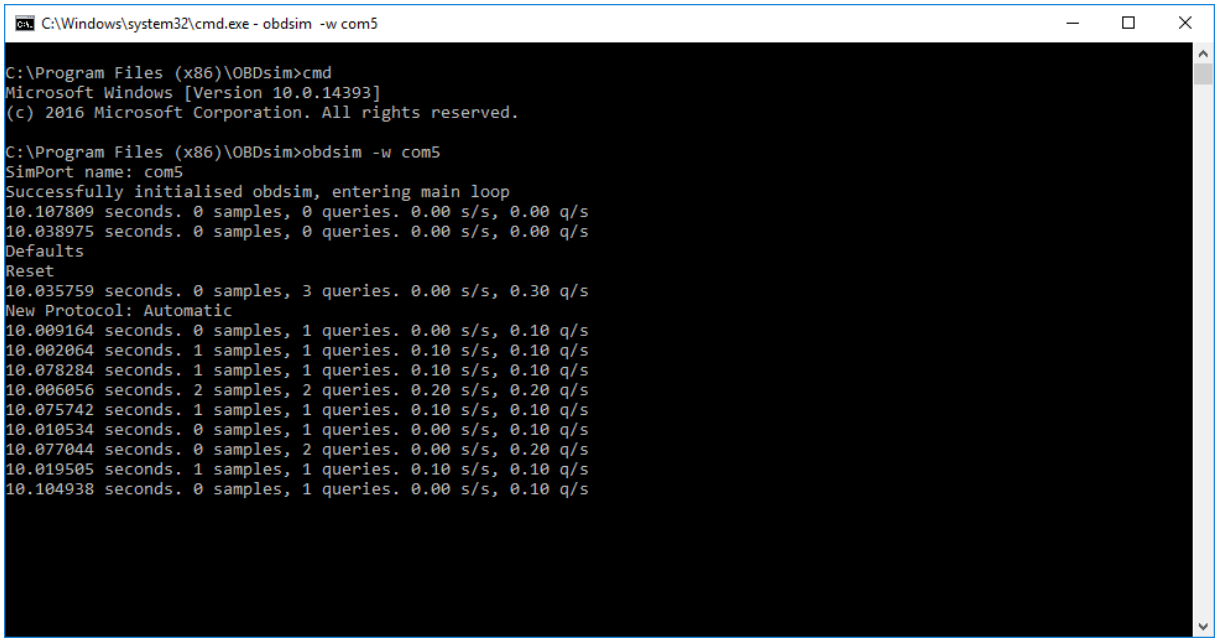

Figura 17: información proporcionada por el simulador *OBDSim* a través de la línea de comandos durante la realización de la prueba.

4.2.2.-Prueba de funcionamiento definitiva de la versión final de la app del proyecto.

Una vez que la programación de la app se dio por terminada, se hizo una prueba final en la que se probaron de una vez todas las características de la app para descartar que hubiese fallos que debieran ser corregidos. El

resultado de la prueba fue completamente satisfactorio, llegándose a la conclusión de que la app podía pasar del entorno de desarrollo al entorno de producción, es decir, que podía compilarse y construirse la versión definitiva de la app (con posibilidad de añadir mejoras en el futuro) a partir del código fuente programado.

En las figuras 18a y 18b se aprecian sendas capturas de pantalla del terminal Android usado en la prueba antes de haber validado la configuración relativa al adaptador o dispositivo Bluetooth OBD2 conectado al vehículo (que en el caso de esta prueba es el adaptador Bluetooth USB conectado al ordenador PC-compatible) y la relativa a los parámetros de conexión con la base de datos en la nube (en lo sucesivo, "los parámetros de configuración de la app"). Por otra parte, en las figuras 19a y 19b se aprecian también sendas capturas de pantalla del terminal Android usado en la prueba después de haber validado los parámetros de configuración de la app. Por último, en la figura 20 se observa la información proporcionada por el simulador *OBDSim* a través de la línea de comandos durante la ejecución de la app del proyecto una vez que han sido validados los parámetros de configuración de la app.

Figura 18a: captura de pantalla de la parte superior de la app antes haber validado los parámetros de configuración.

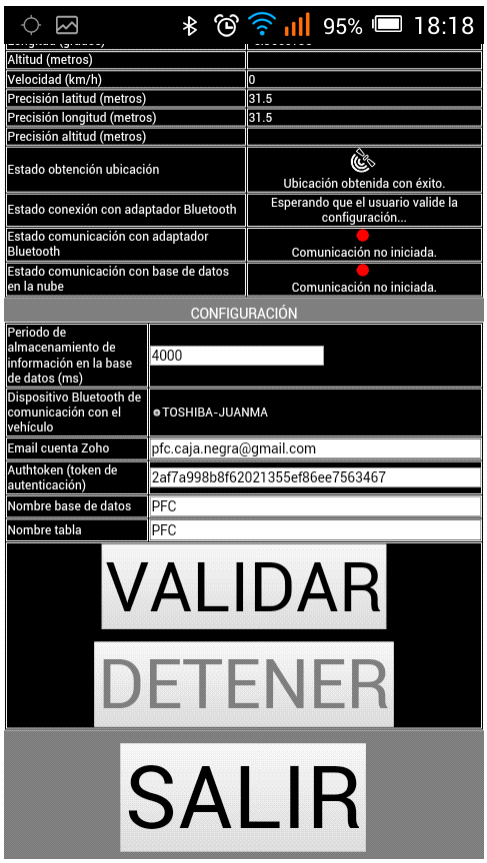

Figura 18b: captura de pantalla de la parte inferior de la app antes de haber validado los parámetros de configuración de la app.

Figura 19a: captura de pantalla de la parte superior de la app después después de haber validado los parámetros de configuración de la app.

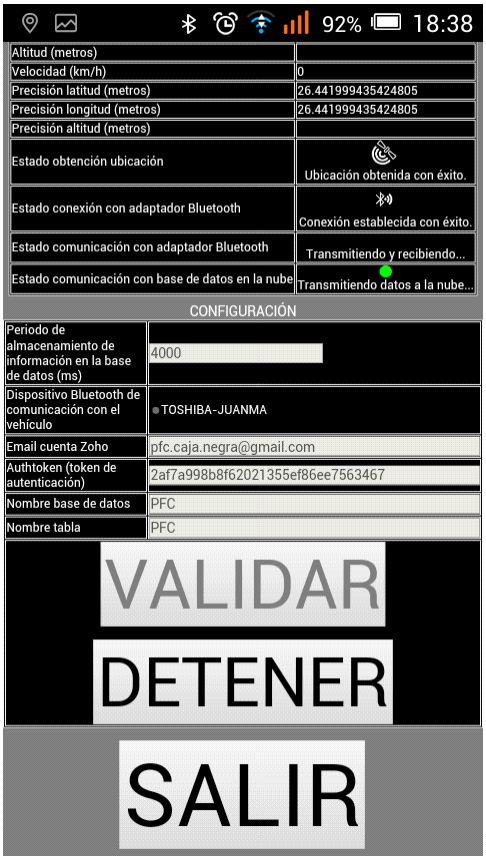

Figura 19b: captura de pantalla de la parte inferior de la app después de haber validado los parámetros de cofiguración de la app.

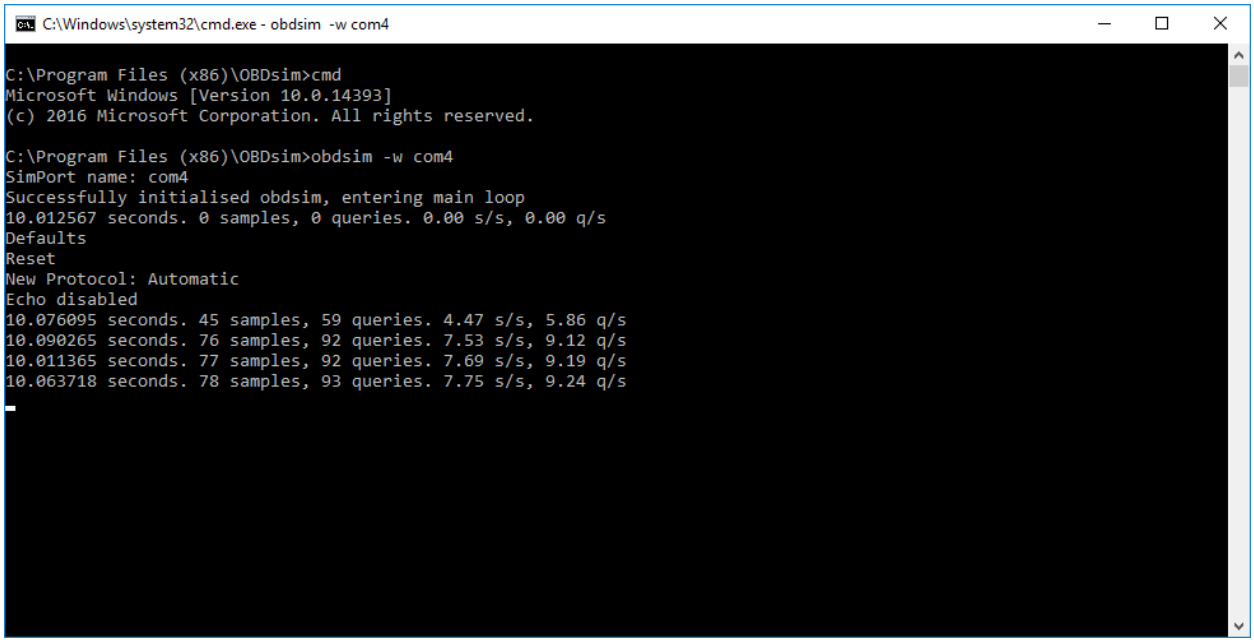

Figura 20: información proporcionada por el simulador *OBDSim* a través de línea de comandos durante la realización de la prueba.

4.2.3.-Prueba de funcionamiento del simulador *OBDSim* y de comunicación entre el terminal Android y el ordenador PC-compatible con la app *Torque Lite*.

La app *Torque Lite*, junto la app *BlueTerm*, fue de gran ayuda de cara a determinar si el origen de algún problema de comunicación por Bluetooth que hubiera entre la app del proyecto y el simulador *OBDSim* residía en la propia app del proyecto o si este era ajeno a ella. La disyuntiva era muy sencilla: si la app *Torque Lite* podía comunicarse por Bluetooth con el simulador *OBDSim* sin ningún problema, pero la app del proyecto no, entonces eso quería decir que el problema de comunicación por Bluetooth con el simulador *OBDSim* estaba en

la propia app del proyecto, que debía revisarse y depurarse detenidamente para encontrar el origen del problema.
Una vez que la configuración externa a la propia app era la adecuada, es decir, una vez que los problemas de configuración o de comunicación ajenos a la propia app estaban solucionados, la app *Torque Lite* siempre pudo establecer comunicación sin problemas con el simulador *OBDSim*. Y no solo eso, sino que además esta app funcionaba de la misma manera que si estuviera comunicándose con la *ECU* de un vehículo real, cuando en realidad estaba comunicándose con el simulador *OBDSim*.
La figura 21 es una captura de la pantalla del terminal Android ejecutando la app *Torque Lite* mientras esta se comunica con el simulador *OBDSim*, mientras que la figura 22 es una captura de pantalla de la línea de comandos del simulador *OBDSim* mientras este se comunica con la app *Torque Lite*.

Figura 21: captura de pantalla del terminal Android durante la ejecución de la app *Torque Lite* mientras se comunica con el simulador *OBDSim*.

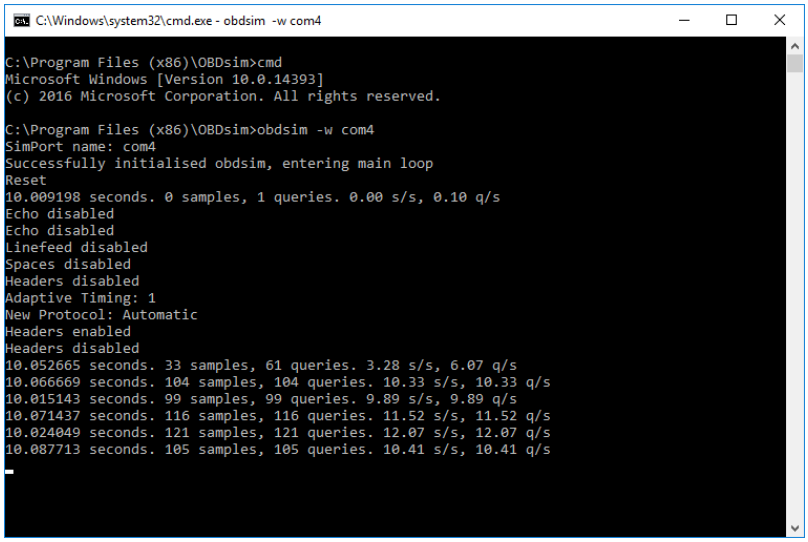

Figura 22: captura de pantalla de la línea de comandos del simulador *OBDSim* mientras se comunica con la app *Torque Lite* del terminal Android.

4.3.-Pruebas llevadas a cabo con un vehículo real.

Distinguimos en este apartado tres categorías o tipos de pruebas: las llevadas a cabo usando la app *BlueTerm*, las llevadas a término usando la app *Torque Lite* y por último las realizadas usando la propia app del proyecto. Evidentemente, a diferencia del apartado anterior, en este apartado el simulador *OBDSim* se ha sustituido por un vehículo real.

4.3.1.-Prueba de envío de comandos básicos con la app *BlueTerm*.

En esta prueba, análoga a la del apartado 4.2.1, se trataba fundamentalmente de comprobar la comunicación por Bluetooth mediante protocolo serie entre el terminal Android y el vehículo real a cuya interfaz OBD2 estaba conectado el dispositivo adaptador Bluetooth OBD2. Mediante la app *BlueTerm* pudieron enviarse al vehículo real comandos básicos de inicialización, así como solicitud de parámetros de funcionamiento del vehículo que la *ECU* del vehículo real supo entender correctamente. El hecho de que la *ECU* del vehículo real entendiera los comandos y las solicitudes de parámetros de funcionamiento del vehículo que se le enviaban era prueba de que todo funcionaba correctamente y según lo previsto, además de un aliciente para programar en la app del proyecto el envío de los mismos comandos de inicialización y de solicitudes de parámetros de funcionamiento del vehículo que se probaron en la app *BlueTerm*. La figura 23 es una captura de la pantalla del terminal Android usado en la prueba, en la que se pueden observar los comandos de inicialización y las solicitudes de parámetros de funcionamiento del vehículo enviados desde la app *BlueTerm* al vehículo real, así como la respuesta recibida procedente del vehículo real en la propia app.

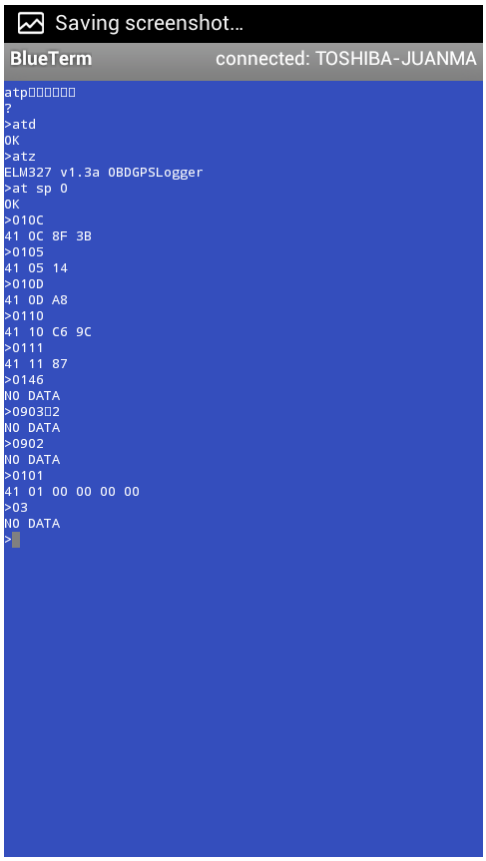

Figura 23: captura de pantalla del terminal Android usado en la prueba con la app *BlueTerm*.

4.3.2.-Prueba de funcionamiento definitiva de la versión final de la app del proyecto.

Una vez que la programación de la app se dio por terminada, se hizo una prueba final en la que se probaron de una vez todas las características de la app para descartar que hubiese fallos que debieran ser corregidos. El resultado de la prueba fue completamente satisfactorio, llegándose a la conclusión de que la app podía pasar del entorno de desarrollo al entorno de producción, es decir, que podía compilarse y construirse la versión definitiva de la app (con posibilidad de añadir mejoras en el futuro) a partir del código fuente programado.

En las figuras 24a y 24b se aprecian sendas capturas de pantalla del terminal Android usado en la prueba antes de haber validado los parámetros de configuración de la app. Por otra parte, en las figuras 25a y 25b se

aprecian sendas capturas de pantalla del terminal Android usado en la prueba después de haber validado los parámetros de configuración de la app.

Figura 24a: captura de pantalla de la parte superior de la app antes de haber validado los parámetros de configuración.

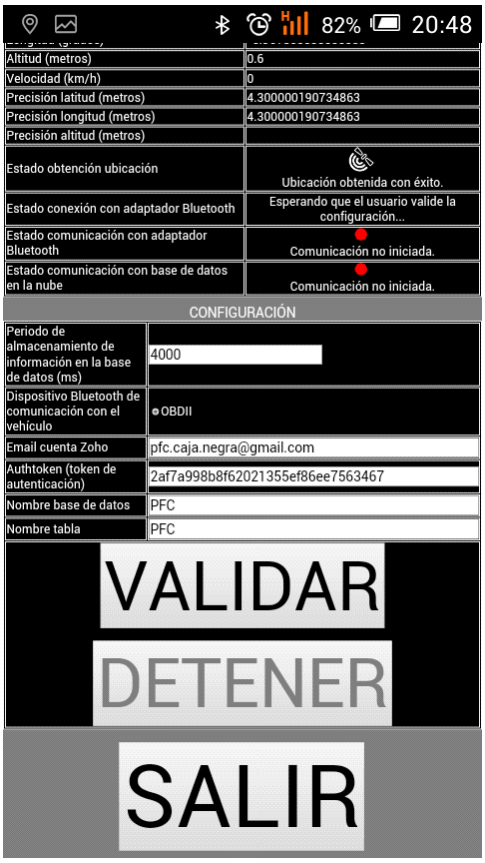

Figura 24b: captura de pantalla de la parte inferior de la app antes de haber validado los parámetros de configuración.

Figura 25a: captura de pantalla de la parte superior de la app después de haber validado los parámetros de configuración.

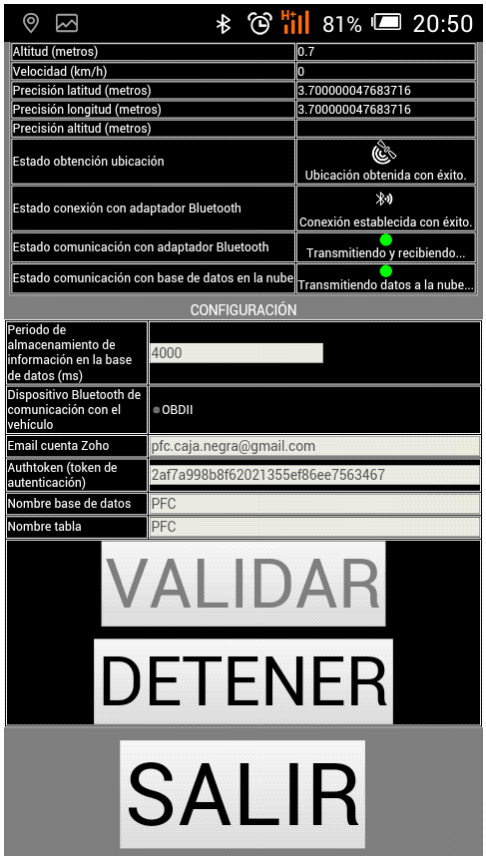

Figura 25b: captura de pantalla de la parte inferior de la app después de haber validado los parámetros de configuración.

4.3.3.-Prueba de comunicación entre el terminal Android y el vehículo real con la app *Torque Lite*.

Al igual que en el apartado análogo 4.2.3, en esta prueba se trataba de llevar a cabo una prueba de comunicación entre el terminal Android y la *ECU* del vehículo, solo que como es lógico, a diferencia de lo que se hizo en la prueba 4.2.3, en este caso la *ECU* era la de un vehículo real en vez de la *ECU* simulada por software del simulador *OBDSim*.

La figura 26 es una captura de pantalla del terminal Android ejecutando la app *Torque Lite* mientras esta se comunica con la ECU del vehículo real.

Figura 26: captura de pantalla del terminal Android ejecutando la app *Torque Lite* mientras esta se comunica con la *ECU* del vehículo real.

5 CONCLUSIONES Y DESARROLLOS FUTUROS

El cerebro humano es como una máquina de acuñar moneda. Si echas en ella metal impuro, obtendrás escoria; si echas oro, obtendrás moneda de ley.

- Santiago Ramón y Cajal -

Para terminar, como colofón final, podríamos decir que, si bien seguramente la app desarrollada no es perfecta, cumple sin embargo con los objetivos para los que ha sido desarrollada, es decir, registra los parámetros obtenidos del terminal Android, los parámetros obtenidos del vehículo a través de la interfaz Bluetooth del terminal y es capaz de ir enviando continuamente los valores de ambos a la cuenta de de la base de datos en le nube configurada previamente por el usuario. Como se ha conmentado en capítulos precedentes, la app se ha desarrollado con el IDE Intel XDK y Apache Cordova. Ambos sistemas en conjunto han permitido desarrollar una app en código fuente de programación web (HTML5, CSS y Javascript) que puede ser compilada para varias plataformas objetivo (Android, iOS, Windows Phone) usando el mismo código fuente de origen (con las adaptaciones necesarias que sean precisas). Esto, que supone una enorme ventaja, dado que no habría que programar la app tantas veces como plataformas destino quieran cubrirse, supone también un gran inconveniente, dado que en ingeniería rara vez algo es gratis. En el caso que nos ocupa, la app se ha programado una única vez para varias plataformas destino, pero esto hace que el tamaño de la app sea considerablemente superior comparado con el caso en el que la app se programa específicamente para cada plataforma o sistema operativo móvil objetivo, además de que los recursos que la app necesita durante su ejecución (memoria RAM y ciclos de CPU principalmente) son también mayores.

Por otra parte, quizá habría sido deseable que el servicio de configuración de base de datos en la nube se hubiera podido configurar con más tipos de servicios de base de datos en la nube además de con Zoho Reports, que es el que se ha elegido y el único que el usuario puede usar con la app. Pero claro, cada servicio de base de datos en la nube se configura de forma diferente al resto, con lo que para que el usuario tenga la posibilidad de elegir entre varios habría que haber programado una serie de botones radio como los que se han programado para listar los dispositivos Bluetooth emparejados con el terminal, además de cambiar los campos del formulario de la tabla de configuración para mostrar unos y ocultar otros en función del servicio de base de datos en la nube que el usuario haya decidido que va a usar. De todas formas, de las muchas posibilidades que se han barajado, Zoho Reports no es tan mala alternativa, ya que no es demasiado difícil de configurar y, sobre todo, permite subir en su versión de prueba gratuita hasta 1000 registros diarios a la base de datos del usuario, algo que otros servicios de base de datos en la nube no ofrecían. En caso de que estos 1000 registros diarios sean insuficientes para el usuario, este siempre puede pagar para salvar las limitaciones de la versión gratuita o de prueba de la cuenta.

Un aspecto positivo a destacar de la app, sin embargo, es que al igual que en el caso del tacógrafo digital (ver apartado 2 del capículo 2) la información que la app obtiene es muy difícil de manipular (al menos sin que el usuario tenga permisos de root o super-administrador). El sistema operativo Android, aún con algunos grandes defectos o puntos negativos, podría considerarse bastante seguro y sería muy complicado para un determinado software de tipo malicioso cambiar los valores de las variables en tiempo de ejecución con el objetivo de manipular fraudulentamente los valores de los parámetros obtenidos por ejemplo del vehículo por la sencilla razón de que el propio sistema operativo no le otorga esos permisos a un usuario que no tenga permisos de root

o de super-administrador. Sin embargo, en el caso de que el usuario sí que tenga permisos de root o de super-administrador, quizá podría usar una cierta app para manipular ciertas variables en tiempo de ejecución de la app y así falsificar por ejemplo los valores de parámetros obtenidos del vehículo. De todas formas, aún en el caso de que se manipulen los valores en tiempo de ejecución de ciertas variables de la app, el fraude estaría solamente ahí, en los valores de los parámetros obtenidos del vehículo o del terminal Android. En ningún caso, aún en el caso de un supuesto fraude, la app pondría en peligro el correcto funcionamiento del motor del vehículo, ni evidentemente como es lógico la conducción, dado que la app solamente lee los parámetros que el vehículo permite. En ningún caso escribe información para por ejemplo cambiar los parámetros de funcionamiento del motor o para "limpiar" los códigos de los errores detectados en el vehículo. En definitiva, la finalidad de la app es solamente informativa.

Del párrafo precedente, y de cara al desarrollo de una futura versión de la app, podría deducirse que debería hacerse algo al respecto para evitar que el usuario del terminal, en caso de que tenga permisos de root o super-administrador, pueda llegar a poder cambiar los valores en tiempo de ejecución de ciertas variables de la app. Una primera opción sería, mediante el uso de un determinado plugin, detectar si el terminal está rooteado, es decir, si una tercera app, contando con el permiso previamente otorgado por el usuario, podría ejecutarse en el terminal con permisos de root o super-administrador y cambiar los valores en tiempo de ejecución de ciertas variables de la app. En caso de que se detectara que el terminal estuviera rooteado, es decir, en caso de que se detectara que una tercera app pudiera llegar a tener permisos de root o super-administrador, la app mostraría un mensaje por pantalla y le daría al usuario dos opciones. La primera opción sería cerrar la app para evitar que una tercera app con permisos de root o super-administrador pueda interferir en el correcto funcionamiento de la app del proyecto fin de carrera. La segunda opción sería continuar la ejecución de la app del proyecto fin de carrera bajo la responsabilidad del usuario, es decir, que en caso de que tuviera lugar una cierta manipulación de las variables en tiempo de ejecución de la app, esta habría sido causada por una tercera app ajena a la app del proyecto fin de carrera, y originada por el hecho de tener el terminal la capacidad de otorgar a ciertas apps (maliciosas o no) permisos de root o super-administrador.

Otro aspecto que quizá también se ha pasado por alto y que sería conveniente tener en cuenta para desarrollos futuros es el hecho de que en caso de que la app se ejecute en modo offline (y que por tanto no envíe a la base de datos en la nube los valores de los parámetros obtenidos del vehículo y del propio terminal Android), una vez que el usuario cierra la app (y salvo que antes previamente haya estado conectado en algún momento a Internet) todos los valores de los parámetros obtenidos del vehículo y del propio terminal Android se perderán. Sería interesante por tanto, en caso de que no se haya hecho copia de seguridad de la base de datos almacenada localmente en la base de datos en la nube, preguntarle al usuario si desea cerrar la app descartando los registros almacenados en la base de datos local o si desea guardarlos en un determinado archivo de la memoria interna o externa del terminal.

ANEXO A: CÓDIGO FUENTE

Contenido del archivo "www/index.html":

```
<!doctype html>
<html>
    <head>
        <title>tittle</title>
        <script>
            window.location='./main.html';
        </script>
    </head>
    <body>
    </body>
</html>
```

Contenido del archivo "www/main.html":

```
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="css/index.css">
        <title>PFC CAJA NEGRA INGENIER&Iacute;A DE TELECOMUNICACI&Oacute;N</title>
    </head>
    <body>
        <table style="margin:auto">
            <th><img src="img/logo_us.gif" height="150" width="150"></th>
            <th><img src="img/logo_etsi.png" height="150" width="150"></th>
        </table>
        <p id="centrado">
            PROYECTO FIN DE CARRERA
            <br>
            CAJA NEGRA PARA TRANSPORTE TERRESTRE
            <br>
            INGENIER&Iacute;A DE TELECOMUNICACI&Oacute;N
            <br>
            ESCUELA T&Eacute;CNICA SUPERIOR DE INGENIER&Iacute;A
            <br>
            UNIVERSIDAD DE SEVILLA
        </p>
        <table id="con_borde">
            <caption id="titulos_tablas">PAR&Aacute;METROS OBTENIDOS DE LA INTERFAZ OBD2 DEL VEH&Iacute;CULO</caption>
            <tr id="con_borde">
                <th id="centrada_con_borde">PAR&Aacute;METRO</th>
                <th id="centrada_con_borde">VALOR</th>
            </tr>
            <tr id="con_borde">
                <td id="con_borde">Revoluciones por minuto (RPM) del motor [0, 16384]</td>
                <td id="con_borde"><div id="RPM_motor"></div></td>
            </tr>
            <tr id="con_borde">
                <td id="con_borde">Posici&oacute;n del acelerador [0, 100]</td>
                <td id="con_borde"><div id="posicion_acelerador"></div></td>
            </tr>

            <tr id="con_borde">
```

```
            <td id="con_borde">Temperatura del motor [-40&#176;C, 215&#176;C]</td>
            <td id="con_borde"><div id="temperatura_motor"></div></td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">Flujo de masa de aire [0 gramos/s, 655 gramos/s]</td>
            <td id="con_borde"><div id="flujo_de_masa_de_aire"></div></td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">Valor del veloc&iacute;metro [0 km/h, 255 km/h]</td>
            <td id="con_borde"><div id="valor_velocimetro"></div></td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">Temperatura ambiente [-40&#176;C, 215&#176;C]</td>
            <td id="con_borde"><div id="temperatura_ambiente"></div></td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">N&uacute;mero de identificaci&oacute;n del veh&iacute;culo (VIN)</td>
            <td id="con_borde"><div id="VIN"></div></td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">N&uacute;mero de errores detectados en el veh&iacute;culo</td>
            <td id="con_borde"><div id="numero_de_errores_detectados_en_vehiculo"></div></td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">C&oacute;digos de error de los errores detectados en el veh&iacute;culo</td>
            <td id="con_borde"><div id="codigos_errores_detectados_en_vehiculo"></div></td>
        </tr>
    </table>
    <br>
    <table id="con_borde">
        <caption id="titulos_tablas">PAR&Aacute;METROS OBTENIDOS DEL TERMINAL ANDROID</caption>
        <tr id="con_borde">
            <th id="centrada_con_borde">PAR&Aacute;METRO</th>
            <th id="centrada_con_borde">VALOR</th>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">Latitud (grados)</td>
            <td id="con_borde"><div id="latitud"></div></td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">Longitud (grados)</td>
            <td id="con_borde"><div id="longitud"></div></td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">Altitud (metros)</td>
            <td id="con_borde"><div id="altitud"></div></td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">Velocidad (km/h)</td>
            <td id="con_borde"><div id="velocidad"></div></td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">Precisi&oacute;n latitud (metros)</td>
            <td id="con_borde"><div id="precision_latitud"></div></td>
        </tr>
        <tr id="con_borde">
```

```
            <td id="con_borde">Precisi&oacute;n longitud (metros)</td>
            <td id="con_borde"><div id="precision_longitud"></div></td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">Precisi&oacute;n altitud (metros)</td>
            <td id="con_borde"><div id="precision_altitud"></div></td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">Estado obtenci&oacute;n ubicaci&oacute;n</td>
            <td id="con_borde">
                <div id="estado_obtencion_ubicacion"></div>
            </td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">Estado conexi&oacute;n con adaptador Bluetooth</td>
            <td id="con_borde">
                <div id="estado_conexion_bluetooth">
                    <p align="center">
                        Esperando que el usuario valide la configuraci&oacute;n...
                    </p>
                </div>
            </td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">Estado comunicaci&oacute;n con adaptador Bluetooth</td>
            <td id="con_borde">
                <div id="estado_comunicacion_bluetooth">
                    <p align="center">
                        <img src="img/led_rojo.png">
                        <br>
                        Comunicaci&oacute;n no iniciada.
                    </p>
                </div>
            </td>
        </tr>
        <tr id="con_borde">
            <td id="con_borde">Estado comunicaci&oacute;n con base de datos en la nube</td>
            <td id="con_borde">
                <div id="estado_comunicacion_Internet">
                    <p align="center">
                        <img src="img/led_rojo.png">
                        <br>
                        Comunicaci&oacute;n no iniciada.
                    </p>
                </div>
            </td>
        </tr>
    </table>
    <br>
    <form name="formulario_configuracion" action="javascript:validar_configuracion(document.forms.formulario_configuracion)">
        <table id="con_borde">
            <caption id="titulos_tablas">CONFIGURACI&Oacute;N</caption>
                <tr id="con_borde">
                    <td id="con_borde">Periodo de almacenamiento de informaci&oacute;n en la base de datos (ms)</td>
                    <td id="con_borde">
```

```
                        <input id="configuracion" name="periodo_almacenamiento_bbdd" type="number" required>
                    </td>
                </tr>
                <tr id="con_borde">
                    <td id="con_borde">Dispositivo Bluetooth de comunicaci&oacute;n con el veh&iacute;culo</td>
                    <td id="con_borde">
                        <div id="dispositivos_bluetooth"></div>
                    </td>
                </tr>
                <tr id="con_borde">
                    <td id="con_borde">Email cuenta Zoho</td>
                    <td id="con_borde">
                        <input id="configuracion" name="email_cuenta_Zoho" type="email" size="40" required>
                    </td>
                </tr>
                <tr id="con_borde">
                    <td id="con_borde">Authtoken (token de autenticaci&oacute;n)</td>
                    <td id="con_borde">
                        <input id="configuracion" name="authtoken" type="text" size="40" required>
                    </td>
                </tr>
                <tr id="con_borde">
                    <td id="con_borde">Nombre base de datos</td>
                    <td id="con_borde">
                        <input id="configuracion" name="nombre_bbdd" type="text" size="40" required>
                    </td>
                </tr>
                <tr id="con_borde">
                    <td id="con_borde">Nombre tabla</td>
                    <td id="con_borde">
                        <input id="configuracion" name="nombre_tabla_bbdd" type="text" size="40" required>
                    </td>
                </tr>
                <tr id="con_borde">
                    <td id="con_borde" colspan="2">
                        <div id="botones_validar_y_detener"></div>
                    </td>
                </tr>
            </table>
        </form>
        <!--<p id="centrado">Se recuerda que para que la app funcione correctamente el GPS y el Bluetooth deben estar activados y debe
estar disponible una conexi&oacute;n a Internet v&aacute;lida.</p>-->
        <br>
        <button onclick="cerrarApp()" id="gordo">SALIR</button>
        <br>
        <br>
        <script src="js/script.js"></script>
        <script src="cordova.js"></script>
    </body>
</html>
```

Contenido del archivo "www/css/index.css":

```
@font-face
{
    font-family: "Arial";
    src: url("./fonts/Arial.ttf");
}
```

```
@font-face
{
    font-family: "Helvetica";
    src: url("./fonts/Helvetica.ttf");
}
@font-face
{
    font-family: "HelveticaNeue";
    src: url("./fonts/HelveticaNeue.ttf");
}
@font-face
{
    font-family: "HelveticaNeue-Light";
    src: url("./fonts/HelveticaNeue-Light.ttf");
}
@font-face
{
    font-family: "sans-serif";
    src: url("./fonts/sans-serif.ttf");
}
@font-face
{
    font-family: "Verdana";
    src: url("./fonts/Verdana.ttf");
}
body
{
    -webkit-touch-callout: none;                /* Para impedir que se copien las imágenes */
    -webkit-text-size-adjust: none;             /* Para impedir que el texto se cambie de tamaño */
    -webkit-user-select: none;                  /* Para impedir que se copie el texto */
    background-color: grey;
    font-family: 'HelveticaNeue-Light', 'HelveticaNeue', 'Helvetica', 'Arial', 'sans-serif';
    font-size: 11px;
    height: 100%;
    margin: 0px;
    padding: 0px;
    width: 100%;
    text-align: center;
}
table#con_borde, th#con_borde, td#con_borde, tr#con_borde
{
    width: auto;
    margin: auto;
    text-align: left;
    color: white;
    font-size: 25px;
    background-color: black;
    border: 2px solid white;
}
table#centrada_con_borde, th#centrada_con_borde, td#centrada_con_borde, tr#centrada_con_borde
{
    width: auto;
    margin: auto;
    text-align: center;
    color: white;
```

```
    background-color: black;
    border: 1px solid white;
}
input#configuracion
{
    font: 28px Verdana;
}
select#configuracion
{
    font: 28px Verdana;
}
p#centrado
{
    text-align: center;
    font: bold 28px Verdana;
}
button#gordo
{
    font: 180px Verdana;
}
input#boton_gordo
{
    font: 140px Verdana;
}
caption#titulos_tablas
{
    font-family: 'HelveticaNeue-Light', 'HelveticaNeue', 'Helvetica', 'Arial', 'sans-serif';
    font-size: 28px;
    color: white;
}
```

Contenido del archivo "www/js/script.js":

```
//configuracion_validada: variable que valdrá true si se ha pulsado el botón "VALIDAR" o false en caso contrario.
var configuracion_validada = false;
//configuracion_detenida: variable que valdrá true si se ha pulsado el botón "DETENER" o false en caso contrario.
var configuracion_detenida = false;
//datos_configuracion: variable que albergará los datos o los parámetros relativos a la configuración de la conexión a la base de datos
almacenada en la nube y a la comunicación por Bluetooth con el vehículo.
var datos_configuracion;


//Hacemos que se ejecute la función onDeviceReady cuando el terminal Android esté listo.
document.addEventListener("deviceready", onDeviceReady, false);


//Función cerrarApp: cierra la app cuando se la invoca.
function cerrarApp()
{
    if (navigator.app)
    {
        navigator.app.exitApp();
    }
    else if (navigator.device)
    {
        navigator.device.exitApp();
    }
    else
    {
        window.close();
```

```
    }
}

//Función validar_configuracion: se la invoca cuando se pulsa en el botón "VALIDAR" de la tabla de configuración.
function validar_configuracion(datos)
{
    //Cambiamos el botón "VALIDAR" a deshabilitado y el botón "DETENER" a habilitado (parámetro disabled):
    document.getElementsByName("boton_validar")[0].disabled = true;
    document.getElementsByName("boton_detener")[0].disabled = false;
    //Hacemos que los parámetros de configuración estén disponibles globalmente en cualquier punto del código de la app:
    datos_configuracion = datos;
    //Mandamos una señal a la función comprobar_validacion_o_detencion_configuracion:
    configuracion_validada = true;
}

//Función detener_configuracion: se la invoca cuando se pulsa el botón "DETENER" de la tabla de configuración.
function detener_configuracion()
{
    //Cambiamos el botón "DETENER" a deshabilitado y el botón "VALIDAR" a habilitado (parámetro disabled):
    document.getElementsByName("boton_validar")[0].disabled = false;
    document.getElementsByName("boton_detener")[0].disabled = true;
    configuracion_detenida = true;
}

//Función onDeviceReady: se ejecuta una vez que el dispositivo (mejor dicho, la app) está completamente listo.
function onDeviceReady()
{
    //Variables que guardan los parámetros (las coordenadas) de la ubicación del terminal.
    var latitud;
    var longitud;
    var altitud;
    var precision_latitud;
    var precision_longitud;
    var precision_altitud;
    var velocidad;
    //Variables que guardan los parámetros relativos a la fecha y la hora en las que se obtiene la ubicación del terminal:
    var fecha;
    var hora;
    var dia;
    var mes;
    var agno;
    var dia_de_la_semana;
    var hora_del_dia;
    var minuto;
    var segundo;
    //Variables que guardan los parámetros relativos al vehículo:
    var VIN; //VIN (Vehicle Identification Number) o número de identificación del vehículo.
    var RPM_motor;
    var temperatura_motor;
    var valor_velocimetro;
    var flujo_de_masa_de_aire;
    var posicion_acelerador;
    var temperatura_ambiente; //Usando el simulador no está disponible. Por lo tanto, valor null en principio.
    var numero_errores_detectados_en_vehiculo; //Cero en principio salvo que más adelante se detecte alguno.
    var codigos_errores_detectados_en_vehiculo; //Lista de los códigos de error detectados en el vehículo.
```

```
    //watchLocationId: variable que guarda la identificación del seguimiento o actualización de la ubicación del terminal, que resulta necesaria para dejar de actualizar las coordenadas de la ubicación del terminal cuando esta cambie.

    var watchLocationId;

    //agregar_fila_bbdd_en_la_nube_id: variable que guarda la identificación de la invocación cada 4 segundos de la función agregar_fila_bbdd_en_la_nube para posteriormente poder detenerla si fuera necesario.

    var agregar_fila_bbdd_en_la_nube_id;

    //agregar_fila_bbdd_local_id: variable que guarda la identificación de la invocación cada 4 segundos de la función agregar_fila_bbdd_local para posteriormente poder detenerla si fuera necesario.

    var agregar_fila_bbdd_local_id;

    //gestionar_comunicacion_bluetooth_id: variable que guarda la identificación de la invocación cada cierto tiempo de la función gestionar_comunicacion_bluetooth para posteriormente poder detenerla si fuera necesario.

    var gestionar_comunicacion_bluetooth_id;

    //comunicacion_Internet_interrumpida_id: variable que guarda la identificación de la invocación cada cierto tiempo de la función comunicacion_Internet_interrumpida para posteriormente poder detenerla si fuera necesario.

    var comunicacion_Internet_interrumpida_id;

    //comunicacion_bluetooth_interrumpida_id: variable que guarda la identificación de la invocación de la función comunicación_bluetooth_interrumpida para detenerla antes de que transcurran los 5 segundos y se determine que la comunicación por Bluetooth se ha interrumpido o detenido.

    var comunicacion_bluetooth_interrumpida_id;

    //ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles_id: variable que guarda la identificación de la invocación de la función ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles para detenerla.

    var ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles_id;

    //comprobar_validacion_configuracion_id: variable que guarda la identificación de la invocación de la función comprobar_validacion_configuracion para detenerla cuando sea necesario.

    var comprobar_validacion_o_detencion_configuracion_id;

    //determinar_bluetooth_y_ubicacion_disponibles_id: variable que guarda la identificación de la invocación de la función determinar_bluetooth_y_ubicacion_disponibles para detenerla cuando sea necesario.

    var comprobar_ubicacion_y_bluetooth_habilitados_id;

    //invocacion_programada_a_funcion_agregar_fila_bbdd_en_la_nube: variable que valdrá true si se ha programado de antemano una invocación a la función agregar_fila_bbdd_en_la_nube o false en caso contrario.

    var invocacion_programada_a_funcion_agregar_fila_bbdd_en_la_nube = false;

    //invocacion_programada_a_funcion_agregar_fila_bbdd_local: variable que valdrá true si se ha programado de antemano una invocación a la función agregar_fila_bbdd_local o false en caso contrario.

    var invocacion_programada_a_funcion_agregar_fila_bbdd_local = false;

    //conexion_Internet_disponible: variable que tiene un valor true si el terminal Android puede conectarse a Internet o un valor false si no hay disponible ningún tipo de conexión a Internet en él.

    var conexion_Internet_disponible = false;

    //ubicacion_disponible: variable que valdrá false hasta que la ubicación del terminal no esté disponible para la app.

    var ubicacion_disponible = false;

    //dispositivos_bluetooth_emparejados: lista de los dispositivos Bluetooth emparejados que la función listar_dispotivos_BluetoothExito ha detectado.

    var dispositivos_bluetooth_emparejados;

    //comunicacion_bluetooth_disponible: variable que valdrá false si no hay comunicación por Bluetooth con el vehículo, o true en caso contrario.

    var comunicacion_bluetooth_disponible = false;

    //configuracion_invalida: variable que valdrá true si bien la ubicación no está disponible, si no hay disponible una conexión a Internet válida o bien si el Bluetooth no está habilitado:

    var configuracion_invalida = false;

    //MAC_adaptador_bluetooth: variable que almacena la MAC del adaptador Bluetooth al que se conectará el terminal Android para obtener los datos relativos al vehículo.

    var MAC_adaptador_bluetooth = "";

    //continuar_ejecucion_app: variable cuyo valor es inicialmente false, aunque es el usuario quien realmente debe darle valor true o false una vez que se muestra en pantalla un mensaje de alerta que indica que es necesario que la ubicación del terminal esté disponible así como que Bluetooth esté habilitado.

    var continuar_ejecucion_app = false;

    //interrumpir_comunicacion_bluetooth: variable bandera que servirá para indicarle a la función gestionar_comunicacion_bluetooth que se quiere interrumpir la comunicación por Bluetooth:

    var interrumpir_comunicacion_bluetooth = false;

    //esperando_recepcion_por_bluetooth: variable bandera que sirve para indicar si se está esperando respuesta de algún comando enviado por Bluetooth

    var esperando_recepcion_por_bluetooth = false;

    //led_verde_bluetooth_apagado: variable que indica si el led verde correspondiente a la conexión con el vehículo a través de la interfaz Bluetooth está apagado (valor true) o si está encendido (valor false).

    var led_verde_bluetooth_apagado = true;

    //led_verde_Internet_apagado: variable que indica si el led verde correspondiente a la conexión a Internet del terminal está apagado (valor true) o encendido (valor false).
```

```
    var led_verde_Internet_apagado = true;

    //numero_de_comando: variable usada por la funcion enviar_comando_por_bluetooth para determinar qué comando es el que tiene que enviarle al vehículo.

    var numero_de_comando;

    //buffer_recepcion_bluetooth: variable en la que se almacenan temporalmente los datos recibidos del vehículo por Bluetooth.

    var buffer_recepcion_bluetooth = "";

    //base_de_datos_local: objeto gestor del acceso a la base de datos almacenada localmente en el terminal (database access handle object).

    var base_de_datos_local;

    //base_de_datos_local_vacia: variable que valdrá true si la base de datos almacenada localmente está vacía (si no contiene ningún registro o fila) o false en caso contrario.

    var base_de_datos_local_vacia = true;

    //base_de_datos_local_abierta: variable que valdrá true si la base de datos almacenada localmente está abierta o false en caso contrario:

    var base_de_datos_local_abierta = false;

    //pasando_bbdd_local_a_bbdd_en_la_nube: variable que valdrá true si se está trabajando en el paso de los registros o filas de la base de datos almacenada localmente a la base de datos en la nube.

    var pasando_bbdd_local_a_bbdd_en_la_nube = false;

    //transaccion_bbdd: variable usada para realizar transacciones en la base de datos almacenada localmente.

    var transaccion_bbdd;

    //formulario_configuración: variable que almacenará la referencia al formulario de configuración de los parámetros de la base de datos en la nube y del dispositivo de comunicación por Bluetooth con el vehículo:

    var formulario_configuracion;

    //Variables que guardan los valores de los parámetros de la tabla de configuración:

    var periodo_almacenamiento_bbdd;

    var dispositivo_bluetooth_seleccionado;

    var email_cuenta_Zoho; //Email de la cuenta del servicio de base de datos en la nube.

    var authtoken; //Token de autenticación para poder hacer la conexión a la base de datos almacenada en la nube.

    var nombre_bbdd; //Nombre de la base de datos almacenada en la nube.

    var nombre_tabla_bbdd; //Nombre de la tabla de la base de datos almacenada en la nube.

    //ubicacion_habilitada: variable que valdrá true si el acceso a la ubicación o localización del terminal está habilitado o false en caso contrario. Por defecto se inicializa a false.

    var ubicacion_habilitada;

    //bluetooth_habilitado: variable que valdrá true si la interfaz Bluetooth del terminal está habilitada o false en caso contrario. Por defecto se inicializa a false.

    var bluetooth_habilitado;

    //configuracion_interrumpida: variable que valdrá true en caso de que se haya pulsado el botón "DETENER" o false si se ha pulsado el botón "VALIDAR":

    var configuracion_interrumpida;

    //CONSTANTES:

    //peticion_PID_RPM: variable correspondiente a la petición PID relativa a las revoluciones por minuto (RPM) del motor.

    const peticion_PID_RPM = "010C\u000D\u000A";

    //peticion_PID_temp_motor: variable correspondiente a la petición PID relativa a la temperatura del motor.

    const peticion_PID_temp_motor = "0105\u000D\u000A";

    //peticion_PID_velocidad: variable correspondiente a la petición PID relativa a la velocidad del vehículo.

    const peticion_PID_velocidad = "010D\u000D\u000A";

    //peticion_PID_flujo_masa_aire: variable correspondiente a la petición PID relativa al flujo de masa de aire.

    const peticion_PID_flujo_masa_aire = "0110\u000D\u000A";

    //peticion_PID_posicion_acelerador: variable correspondiente a la peticion PID relativa a la posición del acelerador.

    const peticion_PID_posicion_acelerador = "0111\u000D\u000A";

    //peticion_PID_temperatura_ambiente: variable correspondiente a la petición PID relativa a la temperatura ambiente.

    const peticion_PID_temperatura_ambiente = "0146\u000D\u000A";

    //peticion_PID_VID: variable correspondiente a la petición PID relativa al VID (Vehicle Identification Number o número de identificación del vehículo).

    const peticion_PID_VIN = "0902\u000D\u000A";

    //peticion_PID_numero_de_errores: variable correspondiente a la petición PID relativa al número de errores detectados en el vehículo:

    const peticion_PID_numero_de_errores = "0101\u000D\u000A";

    //peticion_PID_codigos_de_error: variable correspondiente a la peticion PID relativa a los códigos de error de los errores detectados en el vehículo:

    const peticion_PID_codigos_de_error = "03\u000D\u000A";
```

```
    //comando_ATD: variable correspondiente al comando ATD, que sirve para asignar valores por defecto al integrado ELM327.

    const comando_ATD = "ATD\u000D\u000A";

    //comando_ATZ: variable correspondiente al comando ATZ, que sirve para hacerle un reset al integrado ELM327.

    const comando_ATZ = "ATZ\u000D\u000A";

    //comando_AT_E0: variable correspondiente al comando AT E0, que sirve para deshabilitar el eco.

    const comando_AT_E0 = "AT E0\u000D\u000A";

    //comando_AT_L0: variable correspondiente al comando AT L0, que sirve para deshabilitar el salto de línea.

    const comando_AT_L0 = "AT L0\u000D\u000A";

    //comando_AT_S0: variable correspondiente al comando AT S0, que sirve para deshabilitar la impresión de espacios.

    const comando_AT_S0 = "AT S0\u000D\u000A";

    //comando_AT_H0: variable correspondiente al comando AT H0, que sirve para deshabilitar la visibilidad de las cabeceras del protocolo.

    const comando_AT_H0 = "AT H0\u000D\u000A";

    //comando_AT_SP_0: variable correspondiente al comando AT SP 0, que sirve para otorgarle al integrado ELM327 total libertad de elección de protocolo/s de comunicación con el vehículo.

    const comando_AT_SP_0 = "AT SP 0\u000D\u000A";

    const ratio_conversion_velocidad = 3.6; //Ratio de conversión de velocidad en m/s a velocidad en km/h.

    const inicio_codigos_errores_caso_simulador = 5; //Índice de inicio del array del buffer de recepción (caso simulador).

    const inicio_codigos_errores_caso_vehiculo_real = 3; //Índice de inicio del array del buffer de recepción (caso vehículo real).

    const inicio_buffer_recepcion_caso_simulador = 8;

    const inicio_buffer_recepcion_caso_vehiculo_real = 6;

    const bits_en_un_byte = 8; //Número de bits que contiene un byte.

    const byte_nulo = "00000000"; //Byte nulo (byte con todos sus bits a cero).

    const cinco_segundos = 5000; //5 segundos expresados en ms.

    const tres_segundos = 3000; //3 segundos expresados en ms.

    const medio_segundo = 500; //0,5 segundos expresados en ms.

    const un_cuarto_de_segundo = 250; //0,25 segundos expresados en ms.

    //Intervalo de tiempo expresado en milisegundos transcurrido el cual se borrarán las filas de la columna "Valor" de la tabla relativa a los parámetros obtenidos de la interfaz OBD2 del vehículo.

    const time_out_detencion_configuracion = 2000;

    //Inicio y fin del VIN (Vehicle Identification Number) en el buffer de recepción (caso simulador).

    const inicio_VIN_caso_simulador = 8;

    const fin_VIN_caso_simulador = 25;

    //Inicio y fin del VIN (Vehicle Identification Number) en el buffer de recepción (caso vehículo real).

    const inicio_VIN_caso_vehiculo_real = 6;

    const fin_VIN_caso_vehiculo_real = 23;

    //Intervalo de tiempo medido en ms transcurrido el cual se llamará a la función que gestiona la comunicación Bluetooth:

    const intervalo_gestion_comunicacion_bluetooth = 50;

    const numero_elemento_boton_detener = 7; //Número de elemento del formulario del botón "DETENER".


    //Función convertir_ArrayBuffer_a_String: convierte un ArrayBuffer a String

    function convertir_ArrayBuffer_a_String(array_buffer)

    {

        var array_buffer_8bits = new Uint8Array(array_buffer);

        var cadena_string = "";

        for(i=0; i<array_buffer_8bits.length; i++)

        {

            cadena_string += String.fromCharCode(array_buffer_8bits[i]);

        }

        return cadena_string;

    }


    //Función corrige_entero_menor_a_10: añade un cero a la izquierda al número entero menor a 10 que se le pasa como argumento.

    function corrige_entero_menor_a_10(numero)

    {

        var numero_corregido;
```

```
        if(numero == 0)
        {
            numero_corregido = "00";
        }
        else if(numero == 1)
        {
            numero_corregido = "01";
        }
        else if(numero == 2)
        {
            numero_corregido = "02";
        }
        else if(numero == 3)
        {
            numero_corregido = "03";
        }
        else if(numero == 4)
        {
            numero_corregido = "04";
        }
        else if(numero == 5)
        {
            numero_corregido = "05";
        }
        else if(numero == 6)
        {
            numero_corregido = "06";
        }
        else if(numero == 7)
        {
            numero_corregido = "07";
        }
        else if(numero == 8)
        {
            numero_corregido = "08";
        }
        else if(numero == 9)
        {
            numero_corregido = "09";
        }
        else if(numero > 9)
        {
            numero_corregido = numero;
        }
        return numero_corregido;
    }


    //Función dia_de_la_semana_en_formato_String: recibe el dia de la semana en formato numérico (números del 0 al 6) y devuelve el mes
en formato string (formato texto).
    function dia_de_la_semana_en_formato_String(day_of_week)
    {
        var dia_de_la_semana_String;


        if(day_of_week == 0)
        {
```

```
            dia_de_la_semana_String = "Domingo";
        }
        else if(day_of_week == 1)
        {
            dia_de_la_semana_String = "Lunes";
        }
        else if(day_of_week == 2)
        {
            dia_de_la_semana_String = "Martes";
        }
        else if(day_of_week == 3)
        {
            dia_de_la_semana_String = "Mi\u00E9rcoles";
        }
        else if(day_of_week == 4)
        {
            dia_de_la_semana_String = "Jueves";
        }
        else if(day_of_week == 5)
        {
            dia_de_la_semana_String = "Viernes";
        }
        else if(day_of_week == 6)
        {
            dia_de_la_semana_String = "S\u00E1bado";
        }
        return dia_de_la_semana_String;
    }


    //Función mes_en_formato_String: recibe el mes en formato numérico (números del 0 al 11) y retorna el mes en formato string (formato
texto).
    function mes_en_formato_String(numero_de_mes)
    {
        var mes_String;


        if(numero_de_mes == 0)
        {
            mes_String = "Enero";
        }
        if(numero_de_mes == 1)
        {
            mes_String = "Febrero";
        }
        if(numero_de_mes == 2)
        {
            mes_String = "Marzo";
        }
        if(numero_de_mes == 3)
        {
            mes_String = "Abril";
        }
        if(numero_de_mes == 4)
        {
            mes_String = "Mayo";
        }
        if(numero_de_mes == 5)
```

```
        {
            mes_String = "Junio";
        }
        if(numero_de_mes == 6)
        {
            mes_String = "Julio";
        }
        if(numero_de_mes == 7)
        {
            mes_String = "Agosto";
        }
        if(numero_de_mes == 8)
        {
            mes_String = "Septiembre";
        }
        if(numero_de_mes == 9)
        {
            mes_String = "Octubre";
        }
        if(numero_de_mes == 10)
        {
            mes_String = "Noviembre";
        }
        if(numero_de_mes == 11)
        {
            mes_String = "Diciembre";
        }
        return mes_String;
    }


    function cambiar_estado_led_conexion_Internet()
    {
        //Solo cambiamos el estado del led en caso de que la configuración esté validada, es decir, solamente en caso de que estemos realmente enviando información a la base de datos en la nube:
        if(led_verde_Internet_apagado == true && configuracion_interrumpida != true)
        {
            //Encendemos el led verde:
            document.getElementById("estado_comunicacion_Internet").innerHTML = "<p align='center'><img src='img/led_verde.png'><br>Transmitiendo datos a la nube...</p>";
            //Indicamos que hemos encendido el led verde:
            led_verde_Internet_apagado = false;
        }
        //Solo si el led verde está encendido (y no en rojo), entonces lo apagamos. Si no es así no deberíamos hacerlo, ya que en caso de que la configuración haya sido detenida pondríamos el led verde apagado y con el texto "transmitiendo datos a la nube...":
        else if(led_verde_Internet_apagado == false && configuracion_interrumpida != true)
        {
            //Apagamos el led verde:
            document.getElementById("estado_comunicacion_Internet").innerHTML = "<p align='center'><img src='img/led_apagado.png'><br>Transmitiendo datos a la nube...</p>";
            //Indicamos que hemos apagado el led verde:
            led_verde_Internet_apagado = true;
        }
    }


    //Funcion peticion_POST: función encargada de gestionar las peticiones POST que se hagan al servidor de la base de datos en la nube.
    function peticion_POST()
    {
```

```
        //"Limpiamos" la invocación de la función conexion_Internet_interrumpida, puesto que si se ha creado una nueva petición POST eso implica que no se ha perdido la conexión a Internet, y por tanto estamos suponiendo que tampoco se ha perdido la conexión con la base de datos en la nube.

        clearTimeout(comunicacion_Internet_interrumpida_id);

        //Cambiamos el estado del led correspondiente a la conexión a Internet (si está apagado, lo encendemos, y si está encendido, lo apagamos) y programamos una invocación temporizada a la misma función para hacer lo mismo justo una vez que haya transcurrido la mitad del periodo de tiempo de almacenamiento en la base de datos en la nube:

        cambiar_estado_led_conexion_Internet();

        setTimeout(cambiar_estado_led_conexion_Internet, periodo_almacenamiento_bbdd/2);

        //Versiones de activeX que hay que comprobar en Internet Explorer:

        var activexmodes = ["Msxml2.XMLHTTP", "Microsoft.XMLHTTP"];

        //Caso de navegadores con activeX: comprobamos si existe soporte para ActiveXObject en Internet Explorer:

        if (window.ActiveXObject)

        {

            for (i=0; i<activexmodes.length; i++)

            {

                try

                {

                    return new ActiveXObject(activexmodes[i]);

                }

                //En caso de que lo que se haga dentro de "try" genere error habría que gestionarlo dentro de "catch":

                catch(e)

                {

                }

            }

        }

        //Caso de otros navegadores:

        else if (window.XMLHttpRequest)

        {

            return new XMLHttpRequest();

        }

        else

        {

            return false;

        }

        //Por último, programamos la invocación temporizada de la función conexion_Internet_interrumpida, que cambiará el led verde parpadeante a rojo en caso de que transcurrido un intervalo de tiempo igual al doble del periodo de almacenamiento de datos en la base de datos en la nube no se cree una nueva peticion HTTP POST:

        comunicacion_Internet_interrumpida_id = setTimeout(comunicacion_Internet_interrumpida, periodo_almacenamiento_bbdd*2);

    }


    //Función obtener_tipo_conexion_Internet: obtiene el tipo de conexión a Internet que tiene el terminal Android.

    function obtener_tipo_conexion_Internet()

    {

        var estado_conexion = navigator.connection.type;

        var tipo_conexion = {};

        //Los índices entre corchetes están definidos de antemano en el propio plugin:

        tipo_conexion[Connection.UNKNOWN]  = 'conexión de tipo desconocido';

        tipo_conexion[Connection.ETHERNET] = 'conexión tipo Ethernet';

        tipo_conexion[Connection.WIFI]     = 'conexión tipo WiFi';

        tipo_conexion[Connection.CELL_2G]  = 'conexión celular 2G';

        tipo_conexion[Connection.CELL_3G]  = 'conexión celular 3G';

        tipo_conexion[Connection.CELL_4G]  = 'conexión celular 4G';

        tipo_conexion[Connection.CELL]     = 'conexión celular genérica';

        tipo_conexion[Connection.NONE]     = 'sin conexión';

        return tipo_conexion[estado_conexion];

    }
```

```
    //Función agregar_fila_bbdd_en_la_nube: añade una fila a la correspondiente y previamente definida tabla de la base datos en la nube.
    function agregar_fila_bbdd_en_la_nube()
    {
        var url_base;
        var url_agregar_fila;
        var url_authtoken;
        var url;
        var peticion;
        //Parámetros que guardan los valores de las variables que se pretenden enviar en la petición POST codificados correctamente, dado que pueden contener carácteres especiales.
        var valor_VIN; //Relativo al número de identificación del vehículo (VIN).
        var valor_fecha;
        var valor_hora;
        var valor_latitud;
        var valor_longitud;
        var valor_altitud;
        var valor_precision_latitud;
        var valor_precision_longitud;
        var valor_precision_altitud;
        var valor_velocidad;
        var valor_RPM_motor;
        var valor_temperatura_motor;
        var valor_valor_velocimetro;
        var valor_flujo_de_masa_de_aire;
        var valor_posicion_acelerador;
        var valor_temperatura_ambiente;
        var valor_numero_errores_detectados_en_vehiculo;
        var valor_codigos_errores_detectados_en_vehiculo;
        //Variable que se le pasará como argumento a la función encargada de enviar la peticion POST:
        var parametros = "";
        //fecha_y_hora: variable que guardará la fecha y la hora del registro de la fila en la tabla de la base de datos:
        var fecha_y_hora;

        //alert("Funcion agregar_fila_bbdd_en_la_nube");
        //Creamos un nuevo objeto tipo Date (fecha) con la fecha y la hora actuales:
        fecha_y_hora = new Date();
        //Extraemos los diferentes datos de la fecha y la hora de este nuevo objeto Date (fecha) creado:
        dia = fecha_y_hora.getDate();
        //Si el dia del mes es un número menor a 10, le añadimos un cero a la izquierda:
        if(dia < 10)
        {
            dia = corrige_entero_menor_a_10(dia);
        }
        dia_de_la_semana = dia_de_la_semana_en_formato_String(fecha_y_hora.getDay());
        //Si el mes en formato numérico es menor a 10, le añadimos un cero a la izquierda:
        if((fecha_y_hora.getMonth() + 1) < 10)
        {
            mes = corrige_entero_menor_a_10(fecha_y_hora.getMonth() + 1);
        }
        else if((fecha_y_hora.getMonth() + 1) > 9)
        {
            mes = fecha_y_hora.getMonth() + 1;
        }
        agno = fecha_y_hora.getFullYear();
        hora_del_dia = fecha_y_hora.getHours();
```

```
        //Si la hora del día es un número menor a 10, le añadimos un cero a la izquierda para que la hora se represente exactamente como en un reloj digital:

        if(hora_del_dia < 10)

        {

            hora_del_dia = corrige_entero_menor_a_10(hora_del_dia);

        }

        minuto = fecha_y_hora.getMinutes();

        //Si el minuto es menor a 10, le añadimos un cero a la izquierda para que la hora se represente exactamente como en un reloj digital:

        if(minuto < 10)

        {

            minuto = corrige_entero_menor_a_10(minuto);

        }

        segundo = fecha_y_hora.getSeconds();

        //Si el segundo es menor a 10, le añadimos un cero a la izquierda para que la hora se represente exactamente como en un reloj digital:

        if(segundo < 10)

        {

            segundo = corrige_entero_menor_a_10(segundo);

        }

        //Creamos la fecha a partir de los datos obtenidos anteriormente.

        fecha = dia + "-" + mes + "-" + agno;

        //Creamos la hora a partir de los datos obtenidos anteriormente. Se utiliza el carácter "." en vez de el carácter ":" para separar las horas de los minutos y los minutos de los segundos porque el carácter ":" no permite dar formato de fecha u hora en vez de formato de texto plano a los datos enviados al servidor.

        hora = hora_del_dia + ":" + minuto + ":" + segundo;

        //Preparamos la URL de la petición POST:

        url_base = "https://reportsapi.zoho.com/api/" + email_cuenta_Zoho + "/" + nombre_bbdd + "/" + nombre_tabla_bbdd;

        url_agregar_fila = "?ZOHO_ACTION=ADDROW&ZOHO_OUTPUT_FORMAT=XML&ZOHO_ERROR_FORMAT=XML";

        url_authtoken = "&authtoken=" + authtoken;

        url = url_base + url_agregar_fila + url_authtoken + "&ZOHO_API_VERSION=1.0";

        //Creamos la nueva petición POST:

        peticion = new peticion_POST();

        peticion.onreadystatechange = function()

        {

            if (peticion.readyState == 4)

            {

                if (peticion.status == 200 || window.location.href.indexOf("http") == -1)

                {

                    document.getElementById("result").innerHTML = peticion.responseText;

                }

                else

                {

                    alert("Ha ocurrido un error al hacer la petición HTTP.");

                }

            }

        };

        //Comprobamos que ninguno de los parámetros que pretendemos guardar en la base de datos tiene valor undefined o null, ya que en caso de ser así no se podrá guardar con el formato adecuado. En caso de que el parámetro en cuestión tenga un valor distinto de null y distinto de undefined, se añade al texto de los parámetros de la peticion HTTP POST:

        if(VIN != undefined && VIN != null)

        {

            valor_VIN = encodeURIComponent(VIN);

            if(parametros == "")

            {

                parametros += "N\u00FAmero de identificaci\u00F3n del veh\u00EDculo=";

                parametros += valor_VIN;

            }

            else if(parametros != "")
```

```
    {
        parametros += "&N\u00FAmero de identificaci\u00F3n del veh\u00EDculo=";
        parametros += valor_VIN;
    }
}
if(fecha != undefined && fecha != null)
{
    valor_fecha = encodeURIComponent(fecha);
    if(parametros == "")
    {
        parametros += "Fecha=";
        parametros += valor_fecha;
    }
    else if(parametros != "")
    {
        parametros += "&Fecha=";
        parametros += valor_fecha;
    }
}
if(hora != undefined && hora != null)
{
    valor_hora = encodeURIComponent(hora);
    if(parametros == "")
    {
        parametros += "Hora=";
        parametros += valor_hora;
    }
    else if(parametros != "")
    {
        parametros += "&Hora=";
        parametros += valor_hora;
    }
}
if(latitud != undefined && latitud != null)
{
    valor_latitud = encodeURIComponent(latitud);
    if(parametros == "")
    {
        parametros += "Latitud=";
        parametros += valor_latitud;
    }
    else if(parametros != "")
    {
        parametros += "&Latitud=";
        parametros += valor_latitud;
    }
}
if(longitud != undefined && longitud != null)
{
    valor_longitud = encodeURIComponent(longitud);
    if(parametros == "")
    {
        parametros += "Longitud=";
        parametros += valor_longitud;
    }
```

```
        else if(parametros != "")
        {
            parametros += "&Longitud=";
            parametros += valor_longitud;
        }
    }
    if(altitud != undefined && altitud != null)
    {
        valor_altitud = encodeURIComponent(altitud);
        if(parametros == "")
        {
            parametros += "Altitud=";
            parametros += valor_altitud;
        }
        else if(parametros != "")
        {
            parametros += "&Altitud=";
            parametros += valor_altitud;
        }
    }
    if(precision_latitud != undefined && precision_latitud != null)
    {
        valor_precision_latitud = encodeURIComponent(precision_latitud);
        if(parametros == "")
        {
            parametros += "Precisi\u00F3n latitud=";
            parametros += valor_precision_latitud;
        }
        else if(parametros != "")
        {
            parametros += "&Precisi\u00F3n latitud=";
            parametros += valor_precision_latitud;
        }
    }
    if(precision_longitud != undefined && precision_longitud != null)
    {
        valor_precision_longitud = encodeURIComponent(precision_longitud);
        if(parametros == "")
        {
            parametros += "Precisi\u00F3n longitud=";
            parametros += valor_precision_longitud;
        }
        else if(parametros != "")
        {
            parametros += "&Precisi\u00F3n longitud=";
            parametros += valor_precision_longitud;
        }
    }
    if(precision_altitud != undefined && precision_altitud != null)
    {
        valor_precision_altitud = encodeURIComponent(precision_altitud);
        if(parametros == "")
        {
            parametros += "Precisi\u00F3n altitud=";
            parametros += valor_precision_altitud;
        }
```

```
    else if(parametros != "")
    {
        parametros += "&Precisi\u00F3n altitud=";
        parametros += valor_precision_altitud;
    }
}
if(velocidad != undefined && velocidad != null)
{
    valor_velocidad = encodeURIComponent(velocidad);
    if(parametros == "")
    {
        parametros += "Velocidad=";
        parametros += valor_velocidad;
    }
    else if(parametros != "")
    {
        parametros += "&Velocidad=";
        parametros += valor_velocidad;
    }
}
if(RPM_motor != undefined && RPM_motor != null)
{
    valor_RPM_motor = encodeURIComponent(RPM_motor);
    if(parametros == "")
    {
        parametros += "RPM motor=";
        parametros += valor_RPM_motor;
    }
    else if(parametros != "")
    {
        parametros += "&RPM motor=";
        parametros += valor_RPM_motor;
    }
}
if(temperatura_motor != undefined && temperatura_motor != null)
{
    valor_temperatura_motor = encodeURIComponent(temperatura_motor);
    if(parametros == "")
    {
        parametros += "Temperatura motor=";
        parametros += valor_temperatura_motor;
    }
    else if(parametros != "")
    {
        parametros += "&Temperatura motor=";
        parametros += valor_temperatura_motor;
    }
}
if(valor_velocimetro != undefined && valor_velocimetro != null)
{
    valor_valor_velocimetro = encodeURIComponent(valor_velocimetro);
    if(parametros == "")
    {
        parametros += "Valor veloc\u00EDmetro=";
        parametros += valor_valor_velocimetro;
```

```
        }
        else if(parametros != "")
        {
            parametros += "&Valor veloc\u00EDmetro=";
            parametros += valor_valor_velocimetro;
        }
    }
    if(flujo_de_masa_de_aire != undefined && flujo_de_masa_de_aire != null)
    {
        valor_flujo_de_masa_de_aire = encodeURIComponent(flujo_de_masa_de_aire);
        if(parametros == "")
        {
            parametros += "Flujo de masa de aire=";
            parametros += valor_flujo_de_masa_de_aire;
        }
        else if(parametros != "")
        {
            parametros += "&Flujo de masa de aire=";
            parametros += valor_flujo_de_masa_de_aire;
        }
    }
    if(posicion_acelerador != undefined && posicion_acelerador != null)
    {
        valor_posicion_acelerador = encodeURIComponent(posicion_acelerador);
        if(parametros == "")
        {
            parametros += "Posici\u00F3n acelerador=";
            parametros += valor_posicion_acelerador;
        }
        else if(parametros != "")
        {
            parametros += "&Posici\u00F3n acelerador=";
            parametros += valor_posicion_acelerador;
        }
    }
    if(temperatura_ambiente != undefined && temperatura_ambiente != null)
    {
        valor_temperatura_ambiente = encodeURIComponent(temperatura_ambiente);
        if(parametros == "")
        {
            parametros += "Temperatura ambiente=";
            parametros += valor_temperatura_ambiente;
        }
        else if(parametros != "")
        {
            parametros += "&Temperatura ambiente=";
            parametros += valor_temperatura_ambiente;
        }
    }
    if(numero_errores_detectados_en_vehiculo != undefined && numero_errores_detectados_en_vehiculo != null)
    {
        valor_numero_errores_detectados_en_vehiculo = encodeURIComponent(numero_errores_detectados_en_vehiculo);
        if(parametros == "")
        {
            parametros += "N\u00FAmero de errores detectados en el veh\u00EDculo=";
            parametros += valor_numero_errores_detectados_en_vehiculo;
```

```
            }
            else if(parametros != "")
            {
                parametros += "&N\u00FAmero de errores detectados en el veh\u00EDculo=";
                parametros += valor_numero_errores_detectados_en_vehiculo;
            }
        }
        if(codigos_errores_detectados_en_vehiculo != undefined && codigos_errores_detectados_en_vehiculo != null)
        {
            valor_codigos_errores_detectados_en_vehiculo = encodeURIComponent(codigos_errores_detectados_en_vehiculo);
            if(parametros == "")
            {
                parametros += "C\u00F3digos de los errores detectados en el veh\u00EDculo=";
                parametros += valor_codigos_errores_detectados_en_vehiculo;
            }
            else if(parametros != "")
            {
                parametros += "&C\u00F3digos de los errores detectados en el veh\u00EDculo=";
                parametros += valor_codigos_errores_detectados_en_vehiculo;
            }
        }
        //Abrimos la petición HTTP creada anteriormente e indicamos que es de tipo POST:
        peticion.open("POST", url, true);
        //Indicamos de qué tipo es el contenido de la petición POST:
        peticion.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
        //Enviamos la petición POST pasando como argumento los parámetros correctamente codificados:
        peticion.send(parametros);
    }

    //Función agregar_fila_bbdd_local: añade una fila a la correspondiente y previamente definida tabla de la base de datos almacenada
localmente en la memoria del terminal Android.
    function agregar_fila_bbdd_local()
    {
        //fecha_y_hora: variable que guardará la fecha y la hora del registro de la fila en la tabla de la base de datos:
        var fecha_y_hora;
        var parametros_INSERT_INTO = "(";
        var valores_INSERT_INTO = "(";
        //alert("Funcion agregar_fila_bbdd_local");
        //Creamos un nuevo objeto tipo Date (fecha) con la fecha y la hora actuales:
        fecha_y_hora = new Date();
        //Extraemos los diferentes datos de la fecha y la hora de este nuevo objeto Date (fecha) creado:
        dia = fecha_y_hora.getDate();
        if(dia < 10)
        {
            dia = corrige_entero_menor_a_10(dia);
        }
        dia_de_la_semana = dia_de_la_semana_en_formato_String(fecha_y_hora.getDay());
        if((fecha_y_hora.getMonth() + 1) < 10)
        {
            mes = corrige_entero_menor_a_10(fecha_y_hora.getMonth() + 1);
        }
        else if((fecha_y_hora.getMonth() + 1) > 9)
        {
            mes = fecha_y_hora.getMonth() + 1;
        }
```

```
        agno = fecha_y_hora.getFullYear();
        hora_del_dia = fecha_y_hora.getHours();
        if(hora_del_dia < 10)
        {
            hora_del_dia = corrige_entero_menor_a_10(hora_del_dia);
        }
        minuto = fecha_y_hora.getMinutes();
        //Si el minuto es menor a 10, le añadimos un cero a la izquierda para que la hora se represente exactamente como en un reloj digital:
        if(minuto < 10)
        {
            minuto = corrige_entero_menor_a_10(minuto);
        }
        segundo = fecha_y_hora.getSeconds();
        //Si el segundo es menor a 10, le añadimos un cero a la izquierda para que la hora se represente exactamente como en un reloj digital:
        if(segundo < 10)
        {
            segundo = corrige_entero_menor_a_10(segundo);
        }
        //Creamos la fecha a partir de los datos obtenidos anteriormente.
        fecha = dia + "-" + mes + "-" + agno;
        //Creamos la hora a partir de los datos obtenidos anteriormente.
        hora = hora_del_dia + ":" + minuto + ":" + segundo;
        //Comprobamos si la base de datos almacenada localmente está abierta y, en caso de que sea así, introducimos en ella una fila con los valores de los parámetros de los que se dispone:
        if(base_de_datos_local != null && base_de_datos_local != undefined && transaccion_bbdd != null && transaccion_bbdd != undefined && base_de_datos_local_abierta == true)
        {
            if(VIN != null && VIN != undefined)
            {
                if(parametros_INSERT_INTO.endsWith("(") == true)
                {
                    parametros_INSERT_INTO += "VIN";
                }
                else
                {
                    parametros_INSERT_INTO += ", VIN";
                }
                if(valores_INSERT_INTO.endsWith("(") == true)
                {
                    valores_INSERT_INTO += "'" + VIN + "'";
                }
                else
                {
                    valores_INSERT_INTO += ", '" + VIN + "'";
                }
            }
            if(fecha != null && fecha != undefined)
            {
                if(parametros_INSERT_INTO.endsWith("(") == true)
                {
                    parametros_INSERT_INTO += "Fecha";
                }
                else
                {
                    parametros_INSERT_INTO += ", Fecha";
```

```
        }
        if(valores_INSERT_INTO.endsWith("(") == true)
        {
            valores_INSERT_INTO += "'" + fecha + "'";
        }
        else
        {
            valores_INSERT_INTO += ", '" + fecha + "'";
        }
    }
    if(hora != null && hora != undefined)
    {
        if(parametros_INSERT_INTO.endsWith("(") == true)
        {
            parametros_INSERT_INTO += "Hora";
        }
        else
        {
            parametros_INSERT_INTO += ", Hora";
        }
        if(valores_INSERT_INTO.endsWith("(") == true)
        {
            valores_INSERT_INTO += "'" + hora + "'";
        }
        else
        {
            valores_INSERT_INTO += ", '" + hora + "'";
        }
    }
    if(latitud != null && latitud != undefined)
    {
       if(parametros_INSERT_INTO.endsWith("(") == true)
        {
            parametros_INSERT_INTO += "Latitud";
        }
        else
        {
            parametros_INSERT_INTO += ", Latitud";
        }
        if(valores_INSERT_INTO.endsWith("(") == true)
        {
            valores_INSERT_INTO += latitud;
        }
        else
        {
            valores_INSERT_INTO += ", " + latitud;
        }
    }
    if(longitud != null && longitud != undefined)
    {
        if(parametros_INSERT_INTO.endsWith("(") == true)
        {
            parametros_INSERT_INTO += "Longitud";
        }
        else
```

```
        {
            parametros_INSERT_INTO += ", Longitud";
        }
        if(valores_INSERT_INTO.endsWith("(") == true)
        {
            valores_INSERT_INTO += longitud;
        }
        else
        {
            valores_INSERT_INTO += ", " + longitud;
        }
    }
    if(altitud != null && altitud != undefined)
    {
        if(parametros_INSERT_INTO.endsWith("(") == true)
        {
            parametros_INSERT_INTO += "Altitud";
        }
        else
        {
            parametros_INSERT_INTO += ", Altitud";
        }
        if(valores_INSERT_INTO.endsWith("(") == true)
        {
            valores_INSERT_INTO += altitud;
        }
        else
        {
            valores_INSERT_INTO += ", " + altitud;
        }
    }
    if(precision_latitud != null && precision_latitud != undefined)
    {
        if(parametros_INSERT_INTO.endsWith("(") == true)
        {
            parametros_INSERT_INTO += "Precision_latitud";
        }
        else
        {
            parametros_INSERT_INTO += ", Precision_latitud";
        }
        if(valores_INSERT_INTO.endsWith("(") == true)
        {
            valores_INSERT_INTO += precision_latitud;
        }
        else
        {
            valores_INSERT_INTO += ", " + precision_latitud;
        }
    }
    if(precision_longitud != null && precision_longitud != undefined)
    {
        if(parametros_INSERT_INTO.endsWith("(") == true)
        {
            parametros_INSERT_INTO += "Precision_longitud";
        }
```

```
    else
    {
        parametros_INSERT_INTO += ", Precision_longitud";
    }
    if(valores_INSERT_INTO.endsWith("(") == true)
    {
        valores_INSERT_INTO += precision_longitud;
    }
    else
    {
        valores_INSERT_INTO += ", " + precision_longitud;
    }
}
if(precision_altitud != null && precision_altitud != undefined)
{
    if(parametros_INSERT_INTO.endsWith("(") == true)
    {
        parametros_INSERT_INTO += "Precision_altitud";
    }
    else
    {
        parametros_INSERT_INTO += ", Precision_altitud";
    }
    if(valores_INSERT_INTO.endsWith("(") == true)
    {
        valores_INSERT_INTO += precision_altitud;
    }
    else
    {
        valores_INSERT_INTO += ", " + precision_altitud;
    }
}
if(velocidad != null && velocidad != undefined)
{
    if(parametros_INSERT_INTO.endsWith("(") == true)
    {
        parametros_INSERT_INTO += "Velocidad";
    }
    else
    {
        parametros_INSERT_INTO += ", Velocidad";
    }
    if(valores_INSERT_INTO.endsWith("(") == true)
    {
        valores_INSERT_INTO += velocidad;
    }
    else
    {
        valores_INSERT_INTO += ", " + velocidad;
    }
}
if(RPM_motor != null && RPM_motor != undefined)
{
    if(parametros_INSERT_INTO.endsWith("(") == true)
    {
```

```
            parametros_INSERT_INTO += "RPM_motor";
        }
        else
        {
            parametros_INSERT_INTO += ", RPM_motor";
        }
        if(valores_INSERT_INTO.endsWith("(") == true)
        {
            valores_INSERT_INTO += RPM_motor;
        }
        else
        {
            valores_INSERT_INTO += ", " + RPM_motor;
        }
    }
    if(temperatura_motor != null && temperatura_motor != undefined)
    {
        if(parametros_INSERT_INTO.endsWith("(") == true)
        {
            parametros_INSERT_INTO += "Temperatura_motor";
        }
        else
        {
            parametros_INSERT_INTO += ", Temperatura_motor";
        }
        if(valores_INSERT_INTO.endsWith("(") == true)
        {
            valores_INSERT_INTO += temperatura_motor;
        }
        else
        {
            valores_INSERT_INTO += ", " + temperatura_motor;
        }
    }
    if(valor_velocimetro != null && valor_velocimetro != undefined)
    {
        if(parametros_INSERT_INTO.endsWith("(") == true)
        {
            parametros_INSERT_INTO += "Valor_velocimetro";
        }
        else
        {
            parametros_INSERT_INTO += ", Valor_velocimetro";
        }
        if(valores_INSERT_INTO.endsWith("(") == true)
        {
            valores_INSERT_INTO += valor_velocimetro;
        }
        else
        {
            valores_INSERT_INTO += ", " + valor_velocimetro;
        }
    }
    if(flujo_de_masa_de_aire != null && flujo_de_masa_de_aire != undefined)
    {
        if(parametros_INSERT_INTO.endsWith("(") == true)
```

```
        {
            parametros_INSERT_INTO += "Flujo_de_masa_de_aire";
        }
        else
        {
            parametros_INSERT_INTO += ", Flujo_de_masa_de_aire";
        }
        if(valores_INSERT_INTO.endsWith("(") == true)
        {
            valores_INSERT_INTO += flujo_de_masa_de_aire;
        }
        else
        {
            valores_INSERT_INTO += ", " + flujo_de_masa_de_aire;
        }
    }
    if(posicion_acelerador != null && posicion_acelerador != undefined)
    {
        if(parametros_INSERT_INTO.endsWith("(") == true)
        {
            parametros_INSERT_INTO += "Posicion_acelerador";
        }
        else
        {
            parametros_INSERT_INTO += ", Posicion_acelerador";
        }
        if(valores_INSERT_INTO.endsWith("(") == true)
        {
            valores_INSERT_INTO += posicion_acelerador;
        }
        else
        {
            valores_INSERT_INTO += ", " + posicion_acelerador;
        }
    }
    if(temperatura_ambiente != null && temperatura_ambiente != undefined)
    {
        if(parametros_INSERT_INTO.endsWith("(") == true)
        {
            parametros_INSERT_INTO += "Temperatura_ambiente";
        }
        else
        {
            parametros_INSERT_INTO += ", Temperatura_ambiente";
        }
        if(valores_INSERT_INTO.endsWith("(") == true)
        {
            valores_INSERT_INTO += temperatura_ambiente;
        }
        else
        {
            valores_INSERT_INTO += ", " + temperatura_ambiente;
        }
    }
    if(numero_errores_detectados_en_vehiculo != null && numero_errores_detectados_en_vehiculo != undefined)
```

```
            {
                if(parametros_INSERT_INTO.endsWith("(") == true)
                {
                    parametros_INSERT_INTO += "Numero_errores_detectados_en_vehiculo";
                }
                else
                {
                   parametros_INSERT_INTO += ", Numero_errores_detectados_en_vehiculo";
                }
                if(valores_INSERT_INTO.endsWith("(") == true)
                {
                    valores_INSERT_INTO += numero_errores_detectados_en_vehiculo;
                }
                else
                {
                    valores_INSERT_INTO += ", " + numero_errores_detectados_en_vehiculo;
                }
            }
            if(codigos_errores_detectados_en_vehiculo != null && codigos_errores_detectados_en_vehiculo != undefined)
            {
                if(parametros_INSERT_INTO.endsWith("(") == true)
                {
                    parametros_INSERT_INTO += "Codigos_errores_detectados_en_vehiculo)";
                }
                else
                {
                   parametros_INSERT_INTO += ", Codigos_errores_detectados_en_vehiculo)";
                }
                if(valores_INSERT_INTO.endsWith("(") == true)
                {
                    valores_INSERT_INTO += "'" + codigos_errores_detectados_en_vehiculo + "')";
                }
                else
                {
                    valores_INSERT_INTO += ", '" + codigos_errores_detectados_en_vehiculo + "')";
                }
            }
            //Comprobamos si el último carácter de las variables parametros_INSERT_INTO y valores_INSERT_INTO es ")", y en caso de que no
sea así añadimos este carácter para cerrar o finalizar el texto de los nombres de los parámetros de la transacción y el de sus valores:
            if(parametros_INSERT_INTO.endsWith(")") == false)
            {
                parametros_INSERT_INTO += ")";
            }
            if(valores_INSERT_INTO.endsWith(")") == false)
            {
                valores_INSERT_INTO += ")";
            }
            transaccion_bbdd.executeSql("INSERT INTO PFC " + parametros_INSERT_INTO + " VALUES " + valores_INSERT_INTO);
            base_de_datos_local_vacia = false;
        }
    }


    //Función comprueba_configuracion_Internet: comprueba si el terminal tiene o no conexión a Internet.
    function comprueba_conexion_Internet()
    {
        var tipo_conexion;
```

```
        tipo_conexion = obtener_tipo_conexion_Internet();
        if(tipo_conexion == "sin conexión")
        {
            conexion_Internet_disponible = false;
        }
        else
        {
            conexion_Internet_disponible = true;
        }
    }

    //Función cerrarAPP: cierra la app cuando se la invoca.
    function cerrarApp()
    {
        if (navigator.app)
        {
            navigator.app.exitApp();
        }
        else if (navigator.device)
        {
            navigator.device.exitApp();
        }
        else
        {
            window.close();
        }
    }

    //Función obtener_ubicacionExito: se la invoca cuando la obtención de la ubicación del terminal resulta exitosa.
    function obtener_ubicacionExito(position)
    {
        //Guardamos los parámetros o las coordenadas de la ubicación del terminal en las variables globales correspondientes:
        latitud = position.coords.latitude;
        longitud = position.coords.longitude;
        altitud = position.coords.altitude;
        precision_latitud = position.coords.accuracy;
        precision_longitud = position.coords.accuracy;
        precision_altitud = position.coords.altitudeAccuracy;
        //Multiplicamos por ratio_conversion_velocidad (3,6) para obtener un valor expresado en km/h en vez de en m/s.
        velocidad = position.coords.speed*ratio_conversion_velocidad;
        //Escribimos las coordenadas o los parámetros de la ubicación del terminal en la correspondiente tabla del documento HTML:
        document.getElementById("latitud").innerHTML = latitud;
        document.getElementById("longitud").innerHTML = longitud;
        document.getElementById("altitud").innerHTML = altitud;
        document.getElementById("precision_latitud").innerHTML = precision_latitud;
        document.getElementById("precision_longitud").innerHTML = precision_longitud;
        document.getElementById("precision_altitud").innerHTML = precision_altitud;
        document.getElementById("velocidad").innerHTML = velocidad;
        //Cambiamos el icono y el texto del estado de la obtención de la ubicación del terminal:
        document.getElementById("estado_obtencion_ubicacion").innerHTML = "<p align='center'><img src='img/icono_GPS_estatico.png'> <br>
Ubicaci&oacute;n obtenida con &eacute;xito.</p>";
        //Si es la primera vez que se obtiene la ubicación se indica que la ubicación ya sí está disponible:
        if(ubicacion_disponible == false)
        {
```

```
            ubicacion_disponible = true;
        }
    }

    //Función actualizar_ubicacionExito: se la invoca cuando el terminal cambia de ubicación y este cambio se puede detectar con éxito.
    function actualizar_ubicacionExito(position)
    {
        obtener_ubicacionExito(position);
    }

    //Función obtener_ubicacionExito: se la invoca cuando la app no puede obtener con éxito la ubicación del terminal.
    function obtener_ubicacionError(error)
    {
        console.log(error);
    }

    //Función actualizar_ubicacionError: se la invoca cuando el terminal cambia de ubicación y este cambio no se puede detectar con éxito.
    function actualizar_ubicacionError(error)
    {
        obtener_ubicacionError(error);
    }

    //Función decodificar_codigos_error_detectados_en_vehiculo: se le pasa el buffer de recepción Bluetooth y devuelve una lista con los códigos de error/es detectados en el vehículo.
    function decodificar_codigos_errores_detectados_en_vehiculo(buffer_recepcion, tipo_vehiculo)
    {
        var codigos_de_error = "";
        //Bytes de los códigos de los tres primeros (y posiblemente últimos) errores codificados como enteros sin signo:
        var error_1 = new Uint8Array(2);
        var error_2 = new Uint8Array(2);
        var error_3 = new Uint8Array(2);
        //Codificación en binario de los bytes de los códigos de los errores.
        var error_1_1_binario = "";
        var error_1_2_binario = "";
        var error_2_1_binario = "";
        var error_2_2_binario = "";
        var error_3_1_binario = "";
        var error_3_2_binario = "";
        var error_binario_temporal = "";

        //Codificamos los bytes de los errores recibidos como enteros sin signo.
        //Caso simulador (\r\n03 11 12 21 22 31 32):
        if(tipo_vehiculo == "simulador")
        {
            error_1[0]         =         parseInt("0x"         +         buffer_recepcion[inicio_codigos_errores_caso_simulador]         +
buffer_recepcion[inicio_codigos_errores_caso_simulador + 1]);
            error_1[1]      =      parseInt("0x"      +      buffer_recepcion[inicio_codigos_errores_caso_simulador      +      3]      +
buffer_recepcion[inicio_codigos_errores_caso_simulador + 4]);
            error_2[0]      =      parseInt("0x"      +      buffer_recepcion[inicio_codigos_errores_caso_simulador      +      6]      +
buffer_recepcion[inicio_codigos_errores_caso_simulador + 7]);
            error_2[1]      =      parseInt("0x"      +      buffer_recepcion[inicio_codigos_errores_caso_simulador      +      9]      +
buffer_recepcion[inicio_codigos_errores_caso_simulador + 10]);
            error_3[0]      =      parseInt("0x"      +      buffer_recepcion[inicio_codigos_errores_caso_simulador      +      12]      +
buffer_recepcion[inicio_codigos_errores_caso_simulador + 13]);
            error_3[1]      =      parseInt("0x"      +      buffer_recepcion[inicio_codigos_errores_caso_simulador      +      15]      +
buffer_recepcion[inicio_codigos_errores_caso_simulador + 16]);
        }
        //Caso vehículo real (03 11 12 21 22 31 32):
```

```
        else if(tipo_vehiculo == "vehículo real")

        {

            error_1[0]        =        parseInt("0x"        +        buffer_recepcion[inicio_codigos_errores_caso_vehiculo_real]        +
buffer_recepcion[inicio_codigos_errores_caso_vehiculo_real + 1]);

            error_1[1]     =     parseInt("0x"     +     buffer_recepcion[inicio_codigos_errores_caso_vehiculo_real     +     3]     +
buffer_recepcion[inicio_codigos_errores_caso_vehiculo_real + 4]);

            error_2[0]     =     parseInt("0x"     +     buffer_recepcion[inicio_codigos_errores_caso_vehiculo_real     +     6]     +
buffer_recepcion[inicio_codigos_errores_caso_vehiculo_real + 7]);

            error_2[1]     =     parseInt("0x"     +     buffer_recepcion[inicio_codigos_errores_caso_vehiculo_real     +     9]     +
buffer_recepcion[inicio_codigos_errores_caso_vehiculo_real + 10]);

            error_3[0]     =     parseInt("0x"     +     buffer_recepcion[inicio_codigos_errores_caso_vehiculo_real     +     12]     +
buffer_recepcion[inicio_codigos_errores_caso_vehiculo_real + 13]);

            error_3[1]     =     parseInt("0x"     +     buffer_recepcion[inicio_codigos_errores_caso_vehiculo_real     +     15]     +
buffer_recepcion[inicio_codigos_errores_caso_vehiculo_real + 16]);

        }

        //Pasamos los enteros sin signo anteriores a codificación binaria:

        error_1_1_binario = error_1[0].toString(2);

        error_1_2_binario = error_1[1].toString(2);

        error_2_1_binario = error_2[0].toString(2);

        error_2_2_binario = error_2[1].toString(2);

        error_3_1_binario = error_3[0].toString(2);

        error_3_2_binario = error_3[1].toString(2);

        //Si la codificación binaria de cada byte ocupa menos de 8 bits, la corregimos añadiendo al principio los ceros que hagan falta:

        while(error_1_1_binario.length < bits_en_un_byte)

        {

            error_binario_temporal = error_1_1_binario;

            error_1_1_binario = "0";

            error_1_1_binario += error_binario_temporal;

        }

        while(error_1_2_binario.length < bits_en_un_byte)

        {

            error_binario_temporal = error_1_2_binario;

            error_1_2_binario = "0";

            error_1_2_binario += error_binario_temporal;

        }

        while(error_2_1_binario.length < bits_en_un_byte)

        {

            error_binario_temporal = error_2_1_binario;

            error_2_1_binario = "0";

            error_2_1_binario += error_binario_temporal;

        }

        while(error_2_2_binario.length < bits_en_un_byte)

        {

            error_binario_temporal = error_2_2_binario;

            error_2_2_binario = "0";

            error_2_2_binario += error_binario_temporal;

        }

        while(error_3_1_binario.length < bits_en_un_byte)

        {

            error_binario_temporal = error_3_1_binario;

            error_3_1_binario = "0";

            error_3_1_binario += error_binario_temporal;

        }

        while(error_3_2_binario.length < bits_en_un_byte)

        {

            error_binario_temporal = error_3_2_binario;

            error_3_2_binario = "0";

            error_3_2_binario += error_binario_temporal;
```

```
}
//En caso de que el primer error devuelto por la ECU no sea nulo, calculamos su código de error:
if(error_1_1_binario != byte_nulo && error_1_2_binario != byte_nulo)
{
    if(error_1_1_binario.substr(0,1) == "00")
    {
        codigos_de_error += "P";
    }
    if(error_1_1_binario.substr(0,1) == "01")
    {
        codigos_de_error += "C";
    }
    if(error_1_1_binario.substr(0,1) == "10")
    {
        codigos_de_error += "B";
    }
    if(error_1_1_binario.substr(0,1) == "11")
    {
        codigos_de_error += "U";
    }
    if(error_1_1_binario.substr(2,3) == "00")
    {
        codigos_de_error += "0";
    }
    if(error_1_1_binario.substr(2,3) == "01")
    {
        codigos_de_error += "1";
    }
    if(error_1_1_binario.substr(2,3) == "10")
    {
        codigos_de_error += "2";
    }
    if(error_1_1_binario.substr(2,3) == "11")
    {
        codigos_de_error += "3";
    }
    if(error_1_1_binario.substr(4,7) == "0000")
    {
        codigos_de_error += "0";
    }
    if(error_1_1_binario.substr(4,7) == "0001")
    {
        codigos_de_error += "1";
    }
    if(error_1_1_binario.substr(4,7) == "0010")
    {
        codigos_de_error += "2";
    }
    if(error_1_1_binario.substr(4,7) == "0011")
    {
        codigos_de_error += "3";
    }
    if(error_1_1_binario.substr(4,7) == "0100")
    {
        codigos_de_error += "4";
    }
```

```
if(error_1_1_binario.substr(4,7) == "0101")
{
    codigos_de_error += "5";
}
if(error_1_1_binario.substr(4,7) == "0110")
{
    codigos_de_error += "6";
}
if(error_1_1_binario.substr(4,7) == "0111")
{
    codigos_de_error += "7";
}
if(error_1_1_binario.substr(4,7) == "1000")
{
    codigos_de_error += "8";
}
if(error_1_1_binario.substr(4,7) == "1001")
{
    codigos_de_error += "9";
}
if(error_1_1_binario.substr(4,7) == "1010")
{
    codigos_de_error += "A";
}
if(error_1_1_binario.substr(4,7) == "1011")
{
    codigos_de_error += "B";
}
if(error_1_1_binario.substr(4,7) == "1100")
{
    codigos_de_error += "C";
}
if(error_1_1_binario.substr(4,7) == "1101")
{
    codigos_de_error += "D";
}
if(error_1_1_binario.substr(4,7) == "1110")
{
    codigos_de_error += "E";
}
if(error_1_1_binario.substr(4,7) == "1111")
{
    codigos_de_error += "F";
}
if(error_1_2_binario.substr(0,3) == "0000")
{
    codigos_de_error += "0";
}
if(error_1_2_binario.substr(0,3) == "0001")
{
    codigos_de_error += "1";
}
if(error_1_2_binario.substr(0,3) == "0010")
{
    codigos_de_error += "2";
```

```
}
if(error_1_2_binario.substr(0,3) == "0011")
{
    codigos_de_error += "3";
}
if(error_1_2_binario.substr(0,3) == "0100")
{
    codigos_de_error += "4";
}
if(error_1_2_binario.substr(0,3) == "0101")
{
    codigos_de_error += "5";
}
if(error_1_2_binario.substr(0,3) == "0110")
{
    codigos_de_error += "6";
}
if(error_1_2_binario.substr(0,3) == "0111")
{
    codigos_de_error += "7";
}
if(error_1_2_binario.substr(0,3) == "1000")
{
    codigos_de_error += "8";
}
if(error_1_2_binario.substr(0,3) == "1001")
{
    codigos_de_error += "9";
}
if(error_1_2_binario.substr(0,3) == "1010")
{
    codigos_de_error += "A";
}
if(error_1_2_binario.substr(0,3) == "1011")
{
    codigos_de_error += "B";
}
if(error_1_2_binario.substr(0,3) == "1100")
{
    codigos_de_error += "C";
}
if(error_1_2_binario.substr(0,3) == "1101")
{
    codigos_de_error += "D";
}
if(error_1_2_binario.substr(0,3) == "1110")
{
    codigos_de_error += "E";
}
if(error_1_2_binario.substr(0,3) == "1111")
{
    codigos_de_error += "F";
}
if(error_1_2_binario.substr(4,7) == "0000")
{
    codigos_de_error += "0";
```

```
}
if(error_1_2_binario.substr(4,7) == "0001")
{
    codigos_de_error += "1";
}
if(error_1_2_binario.substr(4,7) == "0010")
{
    codigos_de_error += "2";
}
if(error_1_2_binario.substr(4,7) == "0011")
{
    codigos_de_error += "3";
}
if(error_1_2_binario.substr(4,7) == "0100")
{
    codigos_de_error += "4";
}
if(error_1_2_binario.substr(4,7) == "0101")
{
    codigos_de_error += "5";
}
if(error_1_2_binario.substr(4,7) == "0110")
{
    codigos_de_error += "6";
}
if(error_1_2_binario.substr(4,7) == "0111")
{
    codigos_de_error += "7";
}
if(error_1_2_binario.substr(4,7) == "1000")
{
    codigos_de_error += "8";
}
if(error_1_2_binario.substr(4,7) == "1001")
{
    codigos_de_error += "9";
}
if(error_1_2_binario.substr(4,7) == "1010")
{
    codigos_de_error += "A";
}
if(error_1_2_binario.substr(4,7) == "1011")
{
    codigos_de_error += "B";
}
if(error_1_2_binario.substr(4,7) == "1100")
{
    codigos_de_error += "C";
}
if(error_1_2_binario.substr(4,7) == "1101")
{
    codigos_de_error += "D";
}
if(error_1_2_binario.substr(4,7) == "1110")
{
```

```
            codigos_de_error += "E";
        }
        if(error_1_2_binario.substr(4,7) == "1111")
        {
            codigos_de_error += "F";
        }
    }
    //En caso de que el segundo error devuelto por la ECU no sea nulo, añadimos un espacio y calculamos su código de error:
    if(error_2_1_binario != byte_nulo && error_2_2_binario != byte_nulo)
    {
        codigos_de_error += " ";
        if(error_2_1_binario.substr(0,1) == "00")
        {
            codigos_de_error += "P";
        }
        if(error_2_1_binario.substr(0,1) == "01")
        {
            codigos_de_error += "C";
        }
        if(error_2_1_binario.substr(0,1) == "10")
        {
            codigos_de_error += "B";
        }
        if(error_2_1_binario.substr(0,1) == "11")
        {
            codigos_de_error += "U";
        }
        if(error_2_1_binario.substr(2,3) == "00")
        {
            codigos_de_error += "0";
        }
        if(error_2_1_binario.substr(2,3) == "01")
        {
            codigos_de_error += "1";
        }
        if(error_2_1_binario.substr(2,3) == "10")
        {
            codigos_de_error += "2";
        }
        if(error_2_1_binario.substr(2,3) == "11")
        {
            codigos_de_error += "3";
        }
        if(error_2_1_binario.substr(4,7) == "0000")
        {
            codigos_de_error += "0";
        }
        if(error_2_1_binario.substr(4,7) == "0001")
        {
            codigos_de_error += "1";
        }
        if(error_2_1_binario.substr(4,7) == "0010")
        {
            codigos_de_error += "2";
        }
        if(error_2_1_binario.substr(4,7) == "0011")
```

```
{
    codigos_de_error += "3";
}
if(error_2_1_binario.substr(4,7) == "0100")
{
    codigos_de_error += "4";
}
if(error_2_1_binario.substr(4,7) == "0101")
{
    codigos_de_error += "5";
}
if(error_2_1_binario.substr(4,7) == "0110")
{
    codigos_de_error += "6";
}
if(error_2_1_binario.substr(4,7) == "0111")
{
    codigos_de_error += "7";
}
if(error_2_1_binario.substr(4,7) == "1000")
{
    codigos_de_error += "8";
}
if(error_2_1_binario.substr(4,7) == "1001")
{
    codigos_de_error += "9";
}
if(error_2_1_binario.substr(4,7) == "1010")
{
    codigos_de_error += "A";
}
if(error_2_1_binario.substr(4,7) == "1011")
{
    codigos_de_error += "B";
}
if(error_2_1_binario.substr(4,7) == "1100")
{
    codigos_de_error += "C";
}
if(error_2_1_binario.substr(4,7) == "1101")
{
    codigos_de_error += "D";
}
if(error_2_1_binario.substr(4,7) == "1110")
{
    codigos_de_error += "E";
}
if(error_2_1_binario.substr(4,7) == "1111")
{
    codigos_de_error += "F";
}
if(error_2_2_binario.substr(0,3) == "0000")
{
    codigos_de_error += "0";
}
```

```
if(error_2_2_binario.substr(0,3) == "0001")
{
    codigos_de_error += "1";
}
if(error_2_2_binario.substr(0,3) == "0010")
{
    codigos_de_error += "2";
}
if(error_2_2_binario.substr(0,3) == "0011")
{
    codigos_de_error += "3";
}
if(error_2_2_binario.substr(0,3) == "0100")
{
    codigos_de_error += "4";
}
if(error_2_2_binario.substr(0,3) == "0101")
{
    codigos_de_error += "5";
}
if(error_2_2_binario.substr(0,3) == "0110")
{
    codigos_de_error += "6";
}
if(error_2_2_binario.substr(0,3) == "0111")
{
    codigos_de_error += "7";
}
if(error_2_2_binario.substr(0,3) == "1000")
{
    codigos_de_error += "8";
}
if(error_2_2_binario.substr(0,3) == "1001")
{
    codigos_de_error += "9";
}
if(error_2_2_binario.substr(0,3) == "1010")
{
    codigos_de_error += "A";
}
if(error_2_2_binario.substr(0,3) == "1011")
{
    codigos_de_error += "B";
}
if(error_2_2_binario.substr(0,3) == "1100")
{
    codigos_de_error += "C";
}
if(error_2_2_binario.substr(0,3) == "1101")
{
    codigos_de_error += "D";
}
if(error_2_2_binario.substr(0,3) == "1110")
{
    codigos_de_error += "E";
}
```

```
if(error_2_2_binario.substr(0,3) == "1111")
{
    codigos_de_error += "F";
}
if(error_2_2_binario.substr(4,7) == "0000")
{
    codigos_de_error += "0";
}
if(error_2_2_binario.substr(4,7) == "0001")
{
    codigos_de_error += "1";
}
if(error_2_2_binario.substr(4,7) == "0010")
{
    codigos_de_error += "2";
}
if(error_2_2_binario.substr(4,7) == "0011")
{
    codigos_de_error += "3";
}
if(error_2_2_binario.substr(4,7) == "0100")
{
    codigos_de_error += "4";
}
if(error_2_2_binario.substr(4,7) == "0101")
{
    codigos_de_error += "5";
}
if(error_2_2_binario.substr(4,7) == "0110")
{
    codigos_de_error += "6";
}
if(error_2_2_binario.substr(4,7) == "0111")
{
    codigos_de_error += "7";
}
if(error_2_2_binario.substr(4,7) == "1000")
{
    codigos_de_error += "8";
}
if(error_2_2_binario.substr(4,7) == "1001")
{
    codigos_de_error += "9";
}
if(error_2_2_binario.substr(4,7) == "1010")
{
    codigos_de_error += "A";
}
if(error_2_2_binario.substr(4,7) == "1011")
{
    codigos_de_error += "B";
}
if(error_2_2_binario.substr(4,7) == "1100")
{
    codigos_de_error += "C";
```

```
    }
    if(error_2_2_binario.substr(4,7) == "1101")
    {
        codigos_de_error += "D";
    }
    if(error_2_2_binario.substr(4,7) == "1110")
    {
        codigos_de_error += "E";
    }
    if(error_2_2_binario.substr(4,7) == "1111")
    {
        codigos_de_error += "F";
    }
}
//En caso de que el tercer error devuelto por la ECU no sea nulo, añadimos un espacio y calculamos su código de error:
if(error_3_1_binario != byte_nulo && error_3_2_binario != byte_nulo)
{
    codigos_de_error += " ";
    if(error_3_1_binario.substr(0,1) == "00")
    {
        codigos_de_error += "P";
    }
    if(error_3_1_binario.substr(0,1) == "01")
    {
        codigos_de_error += "C";
    }
    if(error_3_1_binario.substr(0,1) == "10")
    {
        codigos_de_error += "B";
    }
    if(error_3_1_binario.substr(0,1) == "11")
    {
        codigos_de_error += "U";
    }
    if(error_3_1_binario.substr(2,3) == "00")
    {
        codigos_de_error += "0";
    }
    if(error_3_1_binario.substr(2,3) == "01")
    {
        codigos_de_error += "1";
    }
    if(error_3_1_binario.substr(2,3) == "10")
    {
        codigos_de_error += "2";
    }
    if(error_3_1_binario.substr(2,3) == "11")
    {
        codigos_de_error += "3";
    }
    if(error_3_1_binario.substr(4,7) == "0000")
    {
        codigos_de_error += "0";
    }
    if(error_3_1_binario.substr(4,7) == "0001")
    {
```

```
    codigos_de_error += "1";
}
if(error_3_1_binario.substr(4,7) == "0010")
{
    codigos_de_error += "2";
}
if(error_3_1_binario.substr(4,7) == "0011")
{
    codigos_de_error += "3";
}
if(error_3_1_binario.substr(4,7) == "0100")
{
    codigos_de_error += "4";
}
if(error_3_1_binario.substr(4,7) == "0101")
{
    codigos_de_error += "5";
}
if(error_3_1_binario.substr(4,7) == "0110")
{
    codigos_de_error += "6";
}
if(error_3_1_binario.substr(4,7) == "0111")
{
    codigos_de_error += "7";
}
if(error_3_1_binario.substr(4,7) == "1000")
{
    codigos_de_error += "8";
}
if(error_3_1_binario.substr(4,7) == "1001")
{
    codigos_de_error += "9";
}
if(error_3_1_binario.substr(4,7) == "1010")
{
    codigos_de_error += "A";
}
if(error_3_1_binario.substr(4,7) == "1011")
{
    codigos_de_error += "B";
}
if(error_3_1_binario.substr(4,7) == "1100")
{
    codigos_de_error += "C";
}
if(error_3_1_binario.substr(4,7) == "1101")
{
    codigos_de_error += "D";
}
if(error_3_1_binario.substr(4,7) == "1110")
{
    codigos_de_error += "E";
}
if(error_3_1_binario.substr(4,7) == "1111")
```

```
{
    codigos_de_error += "F";
}
if(error_3_2_binario.substr(0,3) == "0000")
{
    codigos_de_error += "0";
}
if(error_3_2_binario.substr(0,3) == "0001")
{
    codigos_de_error += "1";
}
if(error_3_2_binario.substr(0,3) == "0010")
{
    codigos_de_error += "2";
}
if(error_3_2_binario.substr(0,3) == "0011")
{
    codigos_de_error += "3";
}
if(error_3_2_binario.substr(0,3) == "0100")
{
    codigos_de_error += "4";
}
if(error_3_2_binario.substr(0,3) == "0101")
{
    codigos_de_error += "5";
}
if(error_3_2_binario.substr(0,3) == "0110")
{
    codigos_de_error += "6";
}
if(error_3_2_binario.substr(0,3) == "0111")
{
    codigos_de_error += "7";
}
if(error_3_2_binario.substr(0,3) == "1000")
{
    codigos_de_error += "8";
}
if(error_3_2_binario.substr(0,3) == "1001")
{
    codigos_de_error += "9";
}
if(error_3_2_binario.substr(0,3) == "1010")
{
    codigos_de_error += "A";
}
if(error_3_2_binario.substr(0,3) == "1011")
{
    codigos_de_error += "B";
}
if(error_3_2_binario.substr(0,3) == "1100")
{
    codigos_de_error += "C";
}
if(error_3_2_binario.substr(0,3) == "1101")
```

```
{
    codigos_de_error += "D";
}
if(error_3_2_binario.substr(0,3) == "1110")
{
    codigos_de_error += "E";
}
if(error_3_2_binario.substr(0,3) == "1111")
{
    codigos_de_error += "F";
}
if(error_3_2_binario.substr(4,7) == "0000")
{
    codigos_de_error += "0";
}
if(error_3_2_binario.substr(4,7) == "0001")
{
    codigos_de_error += "1";
}
if(error_3_2_binario.substr(4,7) == "0010")
{
    codigos_de_error += "2";
}
if(error_3_2_binario.substr(4,7) == "0011")
{
    codigos_de_error += "3";
}
if(error_3_2_binario.substr(4,7) == "0100")
{
    codigos_de_error += "4";
}
if(error_3_2_binario.substr(4,7) == "0101")
{
    codigos_de_error += "5";
}
if(error_3_2_binario.substr(4,7) == "0110")
{
    codigos_de_error += "6";
}
if(error_3_2_binario.substr(4,7) == "0111")
{
    codigos_de_error += "7";
}
if(error_3_2_binario.substr(4,7) == "1000")
{
    codigos_de_error += "8";
}
if(error_3_2_binario.substr(4,7) == "1001")
{
    codigos_de_error += "9";
}
if(error_3_2_binario.substr(4,7) == "1010")
{
    codigos_de_error += "A";
}
```

```
            if(error_3_2_binario.substr(4,7) == "1011")
            {
                codigos_de_error += "B";
            }
            if(error_3_2_binario.substr(4,7) == "1100")
            {
                codigos_de_error += "C";
            }
            if(error_3_2_binario.substr(4,7) == "1101")
            {
                codigos_de_error += "D";
            }
            if(error_3_2_binario.substr(4,7) == "1110")
            {
                codigos_de_error += "E";
            }
            if(error_3_2_binario.substr(4,7) == "1111")
            {
                codigos_de_error += "F";
            }
        }
        //Devolvemos la variable con los códigos de error:
        return codigos_de_error;
    }

    //Función conexion_bluetooth_interrumpida: provoca que se informe por pantalla de que la comunicación por Bluetooth se ha interrumpido mostrando el led rojo en vez de el led verde parpadeando:
    function comunicacion_bluetooth_interrumpida()
    {
        //Indicamos al resto de la app que la comunicación con el vehículo por Bluetooth ya no está disponible:
        comunicacion_bluetooth_disponible = false;
        //Modificamos el documento HTML para cambiar el LED verde parpadeante por el LED rojo, e indicamos que la comunicación por Bluetooth se ha detenido o interrumpido:
        document.getElementById("estado_comunicacion_bluetooth").innerHTML = "<p align='center'><img src='img/led_rojo.png'><br>Comunicaci&oacute;n interrumpida.</p>";
    }

    //Función enviar_bluetoothExito (opcional): se la invoca cuando el envio de información por Bluetooth tiene éxito.
    function enviar_bluetoothExito()
    {
        //"Limpiamos" la invocación temporizada de la función comunicacion_bluetooth_interrumpida, dado que hemos enviado algo por Bluetooth antes de que hayan transcurrido 5 segundos:
        clearTimeout(comunicacion_bluetooth_interrumpida_id);
        //Si el led verde está apagado, lo encendemos, y si está encendido lo apagamos:
        if(led_verde_bluetooth_apagado == true)
        {
            //Encendemos el led verde:
            document.getElementById("estado_comunicacion_bluetooth").innerHTML = "<p align='center'><img src='img/led_verde.png'><br>Transmitiendo y recibiendo...</p>";
            //Indicamos que hemos encendido el led verde:
            led_verde_bluetooth_apagado = false;
        }
        else if(led_verde_bluetooth_apagado == false)
        {
            //Apagamos el led verde:
            document.getElementById("estado_comunicacion_bluetooth").innerHTML = "<p align='center'><img src='img/led_apagado.png'><br>Transmitiendo y recibiendo...</p>";
            //Indicamos que hemos apagado el led verde:
```

```
            led_verde_bluetooth_apagado = true;

        }

        //Volvemos a invocar con un temporizador de 5 segundos a la función comunicacion_bluetooth_interrumpida: si no se "limpia" la invocación temporizada de esta función antes de que transcurran 5 segundos la app informará de que la comunicación por Bluetooth ha quedado interrumpida.

        comunicacion_bluetooth_interrumpida_id = setTimeout(comunicacion_bluetooth_interrumpida, cinco_segundos);

    }


    //Función recibir_bluetoothExito: se la invoca cuando se recibe información por Bluetooth.

    function recibir_bluetoothExito(datos_recibidos)

    {

        //Convertimos los datos recibidos a formato String:

        var datos_string = convertir_ArrayBuffer_a_String(datos_recibidos);

        var datos_Uint8Array = new Uint8Array(datos_recibidos);

        //Bytes de la respuesta a las peticiones PID en formato hexadecimal:

        var primer_byte_datos_respuesta = "";

        var segundo_byte_datos_respuesta = "";

        var tercer_byte_datos_respuesta = "";

        var cuarto_byte_datos_respuesta = "";

        //Bytes de la respuesta a las peticiones PID en formato decimal:

        var A;

        var B;

        var C;

        var D;


        //alert("Se han recibido " + datos_string.length + " car\u00E1cteres de datos.");

        //alert("Los datos recibidos han sido:\n\n" + "Formato String: " + datos_string + "\n" + "Formato decimal: " + datos_Uint8Array);

        //"Limpiamos" la invocación temporizada de la función comunicacion_bluetooth_interrumpida, ya que acabamos de recibir información del vehículo por Bluetooth:

        clearTimeout(comunicacion_bluetooth_interrumpida_id);

        //Si el led verde está apagado, lo encendemos, y si está encendido lo apagamos:

        if(led_verde_bluetooth_apagado == true)

        {

            //Encendemos el led verde:

            document.getElementById("estado_comunicacion_bluetooth").innerHTML = "<p align='center'><img src='img/led_verde.png'><br>Transmitiendo y recibiendo...</p>";

            //Indicamos que hemos encendido el led verde:

            led_verde_bluetooth_apagado = false;

        }

        else if(led_verde_bluetooth_apagado == false)

        {

            //Apagamos el led verde:

            document.getElementById("estado_comunicacion_bluetooth").innerHTML = "<p align='center'><img src='img/led_apagado.png'><br>Transmitiendo y recibiendo...</p>";

            //Indicamos que hemos apagado el led verde:

            led_verde_bluetooth_apagado = true;

        }

        //Añadimos al buffer de recepción los datos que se acaban de recibir en formato String:

        buffer_recepcion_bluetooth += datos_string;

        //Si se recibe la última parte de la respuesta a un determinado comando o petición de PID (carácter '>'), se analiza la respuesta completa desde el principio para determinar a qué comando o petición de PID corresponde:

        if(buffer_recepcion_bluetooth.endsWith(">") == true)

        {

            //Si la respuesta que recibimos es relativa a una petición de PID relativa al VIN (Vehicle Identification Number o número de identificación del vehículo), mostramos este número de identificación del vehículo en la tabla relativa a los parámetros obtenidos de la interfaz OBD2 del vehículo y guardamos su valor en la variable correspondiente.

            //Caso simulador:

            if(buffer_recepcion_bluetooth.startsWith("\r\n49 02") == true)

            {
```

```
                VIN = "";

                //Copiamos el VIN en la variable correspondiente:

                for(i=inicio_VIN_caso_simulador; i<fin_VIN_caso_simulador; i++)

                {

                    VIN += buffer_recepcion_bluetooth[i];

                }

                //Mostramos el valor del VIN en la tabla correspondiente a los parámetros obtenidos de la interfaz OBD2 del vehículo:

                document.getElementById("VIN").innerHTML = VIN;

            }

            //Caso vehículo real:

            if(buffer_recepcion_bluetooth.startsWith("49 02") == true)

            {

                VIN = "";

                //Copiamos el VIN en la variable correspondiente:

                for(i=inicio_VIN_caso_vehiculo_real; i<fin_VIN_caso_vehiculo_real; i++)

                {

                    VIN += buffer_recepcion_bluetooth[i];

                }

                //Mostramos el valor del VIN en la tabla correspondiente a los parámetros obtenidos de la interfaz OBD2 del vehículo:

                document.getElementById("VIN").innerHTML = VIN;

            }

            //Si la respuesta que recibimos es relativa a una petición PID relativa a la/s luces que indican errores detectados en el
vehículo (MIL), indicamos que se han encontrado errores en el vehículo y su número.

            //Caso simulador:

            if(buffer_recepcion_bluetooth.startsWith("\r\n41 01") == true)

            {

                primer_byte_datos_respuesta      =      buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador)      +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador + 1);

                segundo_byte_datos_respuesta    =    buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador    +    3)    +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador + 4);

                tercer_byte_datos_respuesta    =    buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador    +    6)    +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador + 7);

                cuarto_byte_datos_respuesta    =    buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador    +    9)    +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador + 10);

                A = parseInt("0x" + primer_byte_datos_respuesta);

                B = parseInt("0x" + segundo_byte_datos_respuesta);

                C = parseInt("0x" + tercer_byte_datos_respuesta);

                D = parseInt("0x" + cuarto_byte_datos_respuesta);

                //Si se han encontrado errores en el vehículo, guardamos en la variable correspondiente al número de errores encontrados
en el vehículo el número de errores encontrados en el vehículo:

                if(A != 0)

                {

                    //Si el número de errores no es cero, entonces este es igual a A - 128:

                    numero_errores_detectados_en_vehiculo = A - 128;

                    document.getElementById("numero_de_errores_detectados_en_vehiculo").innerHTML                                    =
numero_errores_detectados_en_vehiculo;

                }

                else if(A == 0)

                {

                    //Si el número de errores sí que es cero, se indica en la correspondiente fila de la tabla relativa a los parámetros
obtenidos de la interfaz OBD2 del vehículo:

                    numero_errores_detectados_en_vehiculo = A;

                    document.getElementById("numero_de_errores_detectados_en_vehiculo").innerHTML                                    =
numero_errores_detectados_en_vehiculo;

                }

            }

            //Caso vehículo real:

            if(buffer_recepcion_bluetooth.startsWith("41 01") == true)

            {

                primer_byte_datos_respuesta     =     buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real)     +
```

```
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real + 1);

                segundo_byte_datos_respuesta  =  buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real  +  3)  +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real + 4);

                tercer_byte_datos_respuesta  =  buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real  +  6)  +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real + 7);

                cuarto_byte_datos_respuesta  =  buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real  +  9)  +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real + 10);

                A = parseInt("0x" + primer_byte_datos_respuesta);

                B = parseInt("0x" + segundo_byte_datos_respuesta);

                C = parseInt("0x" + tercer_byte_datos_respuesta);

                D = parseInt("0x" + cuarto_byte_datos_respuesta);

                //Si se han encontrado errores en el vehículo, guardamos en la variable correspondiente al número de errores encontrados
en el vehículo el número de errores encontrados en el vehículo:

                if(A != 0)

                {

                    //Si el número de errores no es cero, entonces este es igual a A - 128:

                    numero_errores_detectados_en_vehiculo = A - 128;

                    document.getElementById("numero_de_errores_detectados_en_vehiculo").innerHTML                                    =
numero_errores_detectados_en_vehiculo;

                }

                else if(A == 0)

                {

                    //Si el número de errores sí que es cero, se indica en la correspondiente fila de la tabla relativa a los parámetros
obtenidos de la interfaz OBD2 del vehículo:

                    numero_errores_detectados_en_vehiculo = A;

                    document.getElementById("numero_de_errores_detectados_en_vehiculo").innerHTML                                    =
numero_errores_detectados_en_vehiculo;

                }

            }

            //Si la respuesta que recibimos es relativa a una petición PID relativa a la determinación de los códigos de error detectados
en el vehículo, indicamos cuáles han sido los códigos de los errores detectados en el vehículo.

            //Caso simulador:

            if(buffer_recepcion_bluetooth.startsWith("\r\n43") == true)

            {

                codigos_errores_detectados_en_vehiculo  =  decodificar_codigos_errores_detectados_en_vehiculo(buffer_recepcion_bluetooth,
"simulador");

                //Escribimos los códigos de error/es detectado/s en el vehículo en la tabla relativa a los parámetros obtenidos de la
interfaz OBD2 del vehículo:

                document.getElementById("codigos_errores_detectados_en_vehiculo").innerHTML = codigos_errores_detectados_en_vehiculo;

            }

            //Caso vehículo real:

            if(buffer_recepcion_bluetooth.startsWith("43") == true)

            {

                codigos_errores_detectados_en_vehiculo  =  decodificar_codigos_errores_detectados_en_vehiculo(buffer_recepcion_bluetooth,
"vehículo real");

                //Escribimos los códigos de error/es detectado/s en el vehículo en la tabla relativa a los parámetros obtenidos de la
interfaz OBD2 del vehículo:

                document.getElementById("codigos_errores_detectados_en_vehiculo").innerHTML = codigos_errores_detectados_en_vehiculo;

            }

            //Si la respuesta que recibimos indica que no hay datos para la petición de PID enviada, podemos decidir si hacer algo al
respecto...

            //Caso simulador:

            if(buffer_recepcion_bluetooth.startsWith("\r\nNO DATA") == true)

            {

                //No hay datos para la petición de PID enviada.

                //alert("No hay datos para la peticion de PID enviada: NO DATA");

            }

            //Caso vehículo real:

            if(buffer_recepcion_bluetooth.startsWith("NO DATA") == true)

            {

                //No hay datos para la petición de PID enviada.
```

```
                //alert("No hay datos para la peticion de PID enviada: NO DATA");

            }

            //Si la respuesta que recibimos es relativa a una petición PID sobre las RPM del motor, calculamos el valor de este parámetro
a partir de los dos bytes recibidos.

            //Caso simulador:

            if(buffer_recepcion_bluetooth.startsWith("\r\n41 0C") == true)

            {

                primer_byte_datos_respuesta      =      buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador)      +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador + 1);

                segundo_byte_datos_respuesta    =    buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador    +    3)    +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador + 4);

                A = parseInt("0x" + primer_byte_datos_respuesta);

                B = parseInt("0x" + segundo_byte_datos_respuesta);

                RPM_motor = (256*A+B)/4;

                //alert("Las RPM del motor son: " + RPM_motor);

                document.getElementById("RPM_motor").innerHTML = RPM_motor;

            }

            //Caso vehículo real:

            if(buffer_recepcion_bluetooth.startsWith("41 0C") == true)

            {

                primer_byte_datos_respuesta     =     buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real)     +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real + 1);

                segundo_byte_datos_respuesta   =   buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real   +   3)   +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real + 4);

                A = parseInt("0x" + primer_byte_datos_respuesta);

                B = parseInt("0x" + segundo_byte_datos_respuesta);

                RPM_motor = (256*A+B)/4;

                //alert("Las RPM del motor son: " + RPM_motor);

                document.getElementById("RPM_motor").innerHTML = RPM_motor;

            }

            //Si la respuesta que recibimos es relativa a una petición PID sobre la temperatura del motor, calculamos el valor de este
parámetro a partir del byte recibido.

            //Caso simulador:

            if(buffer_recepcion_bluetooth.startsWith("\r\n41 05") == true)

            {

                primer_byte_datos_respuesta      =      buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador)      +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador + 1);

                A = parseInt("0x" + primer_byte_datos_respuesta);

                temperatura_motor = A - 40;

                //alert("La temperatura del motor es: " + temperatura_motor);

                document.getElementById("temperatura_motor").innerHTML = temperatura_motor;

            }

            //Caso vehículo real:

            if(buffer_recepcion_bluetooth.startsWith("41 05") == true)

            {

                primer_byte_datos_respuesta     =     buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real)     +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real + 1);

                A = parseInt("0x" + primer_byte_datos_respuesta);

                temperatura_motor = A - 40;

                //alert("La temperatura del motor es: " + temperatura_motor);

                document.getElementById("temperatura_motor").innerHTML = temperatura_motor;

            }

            //Si la respuesta que recibimos es relativa a una petición PID sobre la velocidad del vehículo, calculamos el valor de este
parámetro a partir del byte recibido.

            //Caso simulador:

            if(buffer_recepcion_bluetooth.startsWith("\r\n41 0D") == true)

            {

                primer_byte_datos_respuesta      =      buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador)      +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador + 1);

                A = parseInt("0x" + primer_byte_datos_respuesta);
```

```
                valor_velocimetro = A;

                //alert("La velocidad del vehículo es: " + velocidad_vehiculo);

                document.getElementById("valor_velocimetro").innerHTML = valor_velocimetro;

            }

            //Caso vehículo real:

            if(buffer_recepcion_bluetooth.startsWith("41 0D") == true)

            {

                primer_byte_datos_respuesta     =     buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real)     +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real + 1);

                A = parseInt("0x" + primer_byte_datos_respuesta);

                valor_velocimetro = A;

                //alert("La velocidad del vehículo es: " + velocidad_vehiculo);

                document.getElementById("valor_velocimetro").innerHTML = valor_velocimetro;

            }

            //Si la respuesta que recibimos es relativa a una petición PID sobre el flujo de masa de aire, calculamos el valor de este
parámetro a partir de los dos bytes recibidos.

            //Caso simulador:

            if(buffer_recepcion_bluetooth.startsWith("\r\n41 10") == true)

            {

                primer_byte_datos_respuesta      =      buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador)      +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador + 1);

                segundo_byte_datos_respuesta    =    buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador    +    3)    +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador + 4);

                A = parseInt("0x" + primer_byte_datos_respuesta);

                B = parseInt("0x" + segundo_byte_datos_respuesta);

                flujo_de_masa_de_aire = (256*A+B)/100;

                //alert("El flujo de masa de aire del motor del vehículo es: " + flujo_de_masa_de_aire);

                document.getElementById("flujo_de_masa_de_aire").innerHTML = flujo_de_masa_de_aire;

            }

            //Caso vehículo real:

            if(buffer_recepcion_bluetooth.startsWith("41 10") == true)

            {

                primer_byte_datos_respuesta     =     buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real)     +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real + 1);

                segundo_byte_datos_respuesta   =   buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real   +   3)   +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real + 4);

                A = parseInt("0x" + primer_byte_datos_respuesta);

                B = parseInt("0x" + segundo_byte_datos_respuesta);

                flujo_de_masa_de_aire = (256*A+B)/100;

                //alert("El flujo de masa de aire del motor del vehículo es: " + flujo_de_masa_de_aire);

                document.getElementById("flujo_de_masa_de_aire").innerHTML = flujo_de_masa_de_aire;

            }

            //Si la respuesta que recibimos es relativa a una petición PID sobre la posición del acelerador, calculamos el valor de este
parámetro a partir del byte recibido.

            //Caso simulador:

            if(buffer_recepcion_bluetooth.startsWith("\r\n41 11") == true)

            {

                primer_byte_datos_respuesta      =      buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador)      +
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador + 1);

                A = parseInt("0x" + primer_byte_datos_respuesta);

                //Aproximamos la posición del acelerador al entero más próximo:

                posicion_acelerador = (A*100)/255;

                //alert("La posición del acelerador es: " + posicion_acelerador);

                document.getElementById("posicion_acelerador").innerHTML = posicion_acelerador;

            }

            //Caso vehículo real:

            if(buffer_recepcion_bluetooth.startsWith("41 11") == true)

            {

                primer_byte_datos_respuesta     =     buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real)     +
```

```
buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real + 1);

                A = parseInt("0x" + primer_byte_datos_respuesta);

                //Aproximamos la posición del acelerador al entero más próximo:

                posicion_acelerador = (A*100)/255;

                //alert("La posición del acelerador es: " + posicion_acelerador);

                document.getElementById("posicion_acelerador").innerHTML = posicion_acelerador;

            }

            //Si la respuesta que recibimos es relativa a una petición PID sobre la temperatura ambiente, calculamos el valor de este parámetro a partir del byte recibido.

            //Caso simulador:

            if(buffer_recepcion_bluetooth.startsWith("\r\n41 46") == true)

            {

                primer_byte_datos_respuesta      =      buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador)      + buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_simulador + 1);

                A = parseInt("0x" + primer_byte_datos_respuesta);

                temperatura_ambiente = A - 40;

                document.getElementById("temperatura_ambiente").innerHTML = temperatura_ambiente;

            }

            //Caso vehículo real:

            if(buffer_recepcion_bluetooth.startsWith("41 46") == true)

            {

                primer_byte_datos_respuesta     =     buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real)     + buffer_recepcion_bluetooth.charAt(inicio_buffer_recepcion_caso_vehiculo_real + 1);

                A = parseInt("0x" + primer_byte_datos_respuesta);

                temperatura_ambiente = A - 40;

                document.getElementById("temperatura_ambiente").innerHTML = temperatura_ambiente;

            }

            //Avisamos de que ya no se está esperando ninguna recepción de datos por Bluetooth, dado que se acaba de terminar de procesar la última que ha llegado:

            esperando_recepcion_por_bluetooth = false;

            //Vaciamos el buffer de recepción Bluetooth para la siguiente respuesta a una petición de parámetro del motor del vehículo:

            buffer_recepcion_bluetooth = "";

        }

        //Volvemos a invocar con un temporizador de 5 segundos a la función comunicacion_bluetooth_interrumpida: si antes de que transcurran 5 segundos no se "limpia" esta invocación, la app entenderá que la comunicación por Bluetooth ha quedado interrumpida o detenida e informará de ello al usuario.

        comunicacion_bluetooth_interrumpida_id = setTimeout(comunicacion_bluetooth_interrumpida, cinco_segundos);

    }


    //Función enviar_comando_por_bluetooth: se encarga de enviarle comandos al vehículo a través de la comunicación por Bluetooth.

    function enviar_comando_por_bluetooth()

    {

        if(numero_de_comando == 1)

        {

            bluetoothSerial.write(comando_ATD, enviar_bluetoothExito);

        }

        if(numero_de_comando == 2)

        {

            bluetoothSerial.write(comando_ATZ, enviar_bluetoothExito);

        }

        if(numero_de_comando == 3)

        {

            bluetoothSerial.write(comando_AT_SP_0, enviar_bluetoothExito);

        }

        if(numero_de_comando == 4)

        {

            bluetoothSerial.write(comando_AT_E0, enviar_bluetoothExito);

        }
```

```
    if(numero_de_comando == 5)
    {
        bluetoothSerial.write(peticion_PID_VIN, enviar_bluetoothExito);
    }
    if(numero_de_comando == 6)
    {
        bluetoothSerial.write(peticion_PID_numero_de_errores, enviar_bluetoothExito);
    }
    if(numero_de_comando == 7)
    {
        bluetoothSerial.write(peticion_PID_codigos_de_error, enviar_bluetoothExito);
    }
    //A partir de aquí, las peticiones de PID que se repetirán indefinidamente:
    if(numero_de_comando == 8)
    {
        bluetoothSerial.write(peticion_PID_RPM, enviar_bluetoothExito);
    }
    if(numero_de_comando == 9)
    {
        bluetoothSerial.write(peticion_PID_temp_motor, enviar_bluetoothExito);
    }
    if(numero_de_comando == 10)
    {
        bluetoothSerial.write(peticion_PID_velocidad, enviar_bluetoothExito);
    }
    if(numero_de_comando == 11)
    {
        bluetoothSerial.write(peticion_PID_flujo_masa_aire, enviar_bluetoothExito);
    }
    if(numero_de_comando == 12)
    {
        bluetoothSerial.write(peticion_PID_posicion_acelerador, enviar_bluetoothExito);
    }
    if(numero_de_comando == 13)
    {
        bluetoothSerial.write(peticion_PID_temperatura_ambiente, enviar_bluetoothExito);
    }
    //Preparamos el envío del siguiente comando de la lista.
    numero_de_comando++;
    if(numero_de_comando > 13)
    {
        numero_de_comando = 8;
    }
}

//Función gestionar_comunicacion_bluetooth: gestiona el envío y recepción de datos a través de la conexión Bluetooth.
function gestionar_comunicacion_bluetooth()
{
    if(esperando_recepcion_por_bluetooth == false)
    {
        enviar_comando_por_bluetooth();
        esperando_recepcion_por_bluetooth = true;
    }
}
```

```
    //Función conexion_bluetoothExito: se la invoca cuando la conexión con el dispositivo Bluetooth conectado a la interfaz OBD2 del vehículo tiene éxito.

    function conexion_bluetoothExito()

    {

        //Cambiamos el icono y el texto de la fila correspondiente al estado de la conexión Bluetooth:

        document.getElementById("estado_conexion_bluetooth").innerHTML = "<p align='center'><img src='img/icono_bluetooth_estatico.png'> <br> Conexi&oacute;n establecida con &eacute;xito.</p>";

        //Iniciamos un temporizador de 5 segundos que, si no se desactiva antes de este periodo de tiempo, provocará que se informe de que la comunicación por Bluetooth se ha interrumpido:

        comunicacion_bluetooth_interrumpida_id = setTimeout(comunicacion_bluetooth_interrumpida, cinco_segundos);

        //alert("Funci\u00F3n conexion_bluetoothExito.");

        //Nos subscribimos a la recepción de los diferentes tipos de datos que se pueden recibir por Bluetooth tras enviar los diferentes comandos que se han programado:

        bluetoothSerial.subscribeRawData(recibir_bluetoothExito);

        //Indicamos que la comunicación por Bluetooth ya sí está disponible:

        comunicacion_bluetooth_disponible = true;

        //Ponemos a 1 el número de comando, para que se empiecen a enviar las órdenes al dispositivo Bluetooth desde el principio:

        numero_de_comando = 1;

        //Como queremos conectarnos y todavía no hemos enviado ningún comando por Bluetooth, indicamos que no estamos esperando la recepción de la respuesta a ningún comando:

        esperando_recepcion_por_bluetooth = false;

        //Programamos una llamada cada cierto tiempo a la función gestionar_comunicacion_bluetooth, que será la encargada de gestionar el envío y la recepción de información a través de Bluetooth:

        gestionar_comunicacion_bluetooth_id = setInterval(gestionar_comunicacion_bluetooth, intervalo_gestion_comunicacion_bluetooth);

    }


    //Función conexion_bluetoothError: se la invoca cuando la conexión con el dispositivo Bluetooth conectado a la interfaz OBD2 del vehículo no tiene éxito.

    function conexion_bluetoothError()

    {

        alert("Funci\u00F3n conexion_bluetoothError.");

    }


    //Función listar_dispositivos_BluetoothExito: se la invoca cuando se consiguen listar los dispositivos Bluetooth emparejados con el terminal Android.

    function listar_dispositivos_BluetoothExito(dispositivos)

    {

        //Si existe algún dispositivo Bluetooth externo con el que el terminal Android esté emparejado, incluimos su/s nombre/s en una lista para que el usuario elija el que quiera usar para obtener los parámetros de funcionamiento del vehículo:

        //alert("Valor dispositivos: " + dispositivos);

        if(dispositivos != null && dispositivos != undefined && dispositivos.length > 0)

        {

            dispositivos_bluetooth_emparejados = dispositivos;

            for(i=0; i<dispositivos.length; i++)

            {

                //Si el dispositivo es el primero de la lista lo dejamos marcado de antemano (opción "checked"):

                if(i == 0)

                {

                    document.getElementById("dispositivos_bluetooth").innerHTML     +=     "<input     id='configuracion'     type='radio' name='dispositivo_bluetooth' value='" + i + "' checked>" + dispositivos[i].name + "<br>";

                }

                //En cambio, si el dispositivo no es el primero de la lista, no lo dejamos marcado (no añadimos la opción "checked"):

                else if(i != 0)

                {

                    document.getElementById("dispositivos_bluetooth").innerHTML     +=     "<input     id='configuracion'     type='radio' name='dispositivo_bluetooth' value='" + i + "'>" + dispositivos[i].name + "<br>";

                }

            }

        }

        //En caso contrario, si no existe ningún dispositivo Bluetooth con el que el terminal Android esté emparejado, se lo indicamos al usuario con un mensaje de advertencia y posteriormente cerramos la app. Solamente en caso de que la interfaz Bluetooth del terminal esté habilitada. En caso contrario, no tiene mucho sentido indicar que no existe ningún dispositivo Bluetooth emparejado con el terminal:
```

```
        else
        {
            if(bluetooth_habilitado == true && ubicacion_habilitada == true)
            {
                navigator.notification.alert("No se ha detectado ning\u00FAn dispositivo Bluetooth externo emparejado con el terminal. Sin embargo, para que la app pueda llevar a cabo su cometido correctamente, es necesario que el terminal est\u00E9 emparejado con al menos un dispositivo Bluetooth conectado a la interfaz OBD2 del veh\u00EDculo. Pulse 'Aceptar' para cerrar la app. Posteriormente, empareje el terminal con el dispositivo Bluetooth conectado a la interfaz OBD2 del veh\u00EDculo y vuelva a iniciar la app.", cerrarApp, "Terminal no emparejado", "Aceptar");
            }
        }
    }


    //Función pasar_contenido_bbdd_local_a_bbdd_en_la_nubeError: se la invoca cuando la transacción SQL "SELECT * FROM PFC" da error:
    function pasar_contenido_bbdd_local_a_bbdd_en_la_nubeError(transaccion, error)
    {
        alert("Error al intentar obtener el contenido de la base de datos almacenada localmente: " + error.message);
    }


    //Función pasar_contenido_bbdd_local_a_bbdd_en_la_nubeExito: se la invoca cuando la transacción SQL "SELECT * FROM PFC" tiene éxito:
    function pasar_contenido_bbdd_local_a_bbdd_en_la_nubeExito(transaccion, resultado)
    {
        var peticion; //Variable objeto que servirá para gestionar la peticion POST.
        var parametros; //Variable que guardará el texto codificado de los parámetros de la peticion POST.
        //Variables en las que se guardarán los parámetros que se extraigan de la base de datos almacenada localmente:
        var VIN_bbdd_local;
        var fecha_bbdd_local;
        var hora_bbdd_local;
        var latitud_bbdd_local;
        var longitud_bbdd_local;
        var altitud_bbdd_local;
        var precision_latitud_bbdd_local;
        var precision_longitud_bbdd_local;
        var precision_altitud_bbdd_local;
        var velocidad_bbdd_local;
        var RPM_motor_bbdd_local;
        var temperatura_motor_bbdd_local;
        var valor_velocimetro_bbdd_local;
        var flujo_de_masa_de_aire_bbdd_local;
        var posicion_acelerador_bbdd_local;
        var temperatura_ambiente_bbdd_local;
        var numero_errores_detectados_en_vehiculo_bbdd_local;
        var codigos_errores_detectados_en_vehiculo_bbdd_local;
        //Variables que guardan los valores de los parámetros de la petición POST con los carácteres especiales codificados correctamente para que puedan enviarse en la URL de la petición POST:
        var valor_VIN;
        var valor_fecha;
        var valor_hora;
        var valor_latitud;
        var valor_longitud;
        var valor_altitud;
        var valor_precision_latitud;
        var valor_precision_longitud;
        var valor_precision_altitud;
        var valor_velocidad;
        var valor_RPM_motor;
        var valor_temperatura_motor;
```

```
        var valor_velocimetro;
        var valor_flujo_de_masa_de_aire;
        var valor_posicion_acelerador;
        var valor_temperatura_ambiente;
        var valor_numero_errores_detectados_en_vehiculo;
        var valor_codigos_errores_detectados_en_vehiculo;
        //Variables relativas a la URL de las peticiones POST:
        var url;
        var url_base;
        var url_agregar_fila;
        var url_authtoken;

        //Indicamos al resto de la app que estamos trabajando en pasar el contenido de la base de datos almacenada localmente a la base
de datos en la nube:
        pasando_bbdd_local_a_bbdd_en_la_nube = true;
        //Le damos valor a las variables correspondientes a las peticiones POST que se enviaran al servidor.
        url_base = "https://reportsapi.zoho.com/api/" + email_cuenta_Zoho + "/" + nombre_bbdd + "/" + nombre_tabla_bbdd;
        url_agregar_fila = "?ZOHO_ACTION=ADDROW&ZOHO_OUTPUT_FORMAT=XML&ZOHO_ERROR_FORMAT=XML";
        url_authtoken = "&authtoken=" + authtoken;
        url = url_base + url_agregar_fila + url_authtoken + "&ZOHO_API_VERSION=1.0";
        //Iteramos por todos los elementos (filas) de la transacción SELECT:
        for(i=0; i<resultado.rows.length; i++)
        {
            //"Limpiamos" la variable parametros (en caso de que tenga el valor de una iteración anterior del bucle) o la inicializamos
(en caso de que esta sea la primera iteración del bucle):
            var parametros = "";
            //Extraemos de la fila correspondiente a esta iteración del bucle for los valores de sus campos:
            VIN_bbdd_local = resultado.rows.item(i).VIN;
            fecha_bbdd_local = resultado.rows.item(i).Fecha;
            hora_bbdd_local = resultado.rows.item(i).Hora;
            latitud_bbdd_local = resultado.rows.item(i).Latitud;
            longitud_bbdd_local = resultado.rows.item(i).Longitud;
            altitud_bbdd_local = resultado.rows.item(i).Altitud;
            precision_latitud_bbdd_local = resultado.rows.item(i).Precision_latitud;
            precision_longitud_bbdd_local = resultado.rows.item(i).Precision_longitud;
            precision_altitud_bbdd_local = resultado.rows.item(i).Precision_altitud;
            velocidad_bbdd_local = resultado.rows.item(i).Velocidad;
            RPM_motor_bbdd_local = resultado.rows.item(i).RPM_motor;
            temperatura_motor_bbdd_local = resultado.rows.item(i).Temperatura_motor;
            valor_velocimetro_bbdd_local = resultado.rows.item(i).Valor_velocimetro;
            flujo_de_masa_de_aire_bbdd_local = resultado.rows.item(i).Flujo_de_masa_de_aire;
            posicion_acelerador_bbdd_local = resultado.rows.item(i).Posicion_acelerador;
            temperatura_ambiente_bbdd_local = resultado.rows.item(i).Temperatura_ambiente;
            numero_errores_detectados_en_vehiculo_bbdd_local = resultado.rows.item(i).Numero_errores_detectados_en_vehiculo;
            codigos_errores_detectados_en_vehiculo_bbdd_local = resultado.rows.item(i).Codigos_errores_detectados_en_vehiculo;
            //Creamos una nueva petición POST:
            peticion = new peticion_POST();
            peticion.onreadystatechange = function()
            {
                if (peticion.readyState == 4)
                {
                    if (peticion.status == 200 || window.location.href.indexOf("http") == -1)
                    {
                        document.getElementById("result").innerHTML = peticion.responseText;
                    }
                    else
```

```
                {
                    //alert("Ha ocurrido un error al hacer la petición HTTP.");
                }
            }
        };
        //Comprobamos que ninguno de los parámetros que pretendemos guardar en la base de datos tiene valor undefined o null, ya que en caso de ser así no se podrá guardar con el formato adecuado. En caso de que el parámetro en cuestión tenga un valor distinto de null y distinto de undefined, se añade al texto de los parámetros de la peticion HTTP POST:
        if(VIN_bbdd_local != undefined && VIN_bbdd_local != null)
        {
            valor_VIN = encodeURIComponent(VIN_bbdd_local);
            if(parametros == "")
            {
                parametros += "N\u00FAmero de identificaci\u00F3n del veh\u00EDculo=";
                parametros += valor_VIN;
            }
            else if(parametros != "")
            {
                parametros += "&N\u00FAmero de identificaci\u00F3n del veh\u00EDculo=";
                parametros += valor_VIN;
            }
        }
        if(fecha_bbdd_local != undefined && fecha_bbdd_local != null)
        {
            valor_fecha = encodeURIComponent(fecha_bbdd_local);
            if(parametros == "")
            {
                parametros += "Fecha=";
                parametros += valor_fecha;
            }
            else if(parametros != "")
            {
                parametros += "&Fecha=";
                parametros += valor_fecha;
            }
        }
        if(hora_bbdd_local != undefined && hora_bbdd_local != null)
        {
            valor_hora = encodeURIComponent(hora_bbdd_local);
            if(parametros == "")
            {
                parametros += "Hora=";
                parametros += valor_hora;
            }
            else if(parametros != "")
            {
                parametros += "&Hora=";
                parametros += valor_hora;
            }
        }
        if(latitud_bbdd_local != undefined && latitud_bbdd_local != null)
        {
            valor_latitud = encodeURIComponent(latitud_bbdd_local);
            if(parametros == "")
            {
                parametros += "Latitud=";
```

```
            parametros += valor_latitud;
        }
        else if(parametros != "")
        {
            parametros += "&Latitud=";
            parametros += valor_latitud;
        }
    }
    if(longitud_bbdd_local != undefined && longitud_bbdd_local != null)
    {
        valor_longitud = encodeURIComponent(longitud_bbdd_local);
        if(parametros == "")
        {
            parametros += "Longitud=";
            parametros += valor_longitud;
        }
        else if(parametros != "")
        {
            parametros += "&Longitud=";
            parametros += valor_longitud;
        }
    }
    if(altitud_bbdd_local != undefined && altitud_bbdd_local != null)
    {
        valor_altitud = encodeURIComponent(altitud_bbdd_local);
        if(parametros == "")
        {
            parametros += "Altitud=";
            parametros += valor_altitud;
        }
        else if(parametros != "")
        {
            parametros += "&Altitud=";
            parametros += valor_altitud;
        }
    }
    if(precision_latitud_bbdd_local != undefined && precision_latitud_bbdd_local != null)
    {
        valor_precision_latitud = encodeURIComponent(precision_latitud_bbdd_local);
        if(parametros == "")
        {
            parametros += "Precisi\u00F3n latitud=";
            parametros += valor_precision_latitud;
        }
        else if(parametros != "")
        {
            parametros += "&Precisi\u00F3n latitud=";
            parametros += valor_precision_latitud;
        }
    }
    if(precision_longitud_bbdd_local != undefined && precision_longitud_bbdd_local != null)
    {
        valor_precision_longitud = encodeURIComponent(precision_longitud_bbdd_local);
        if(parametros == "")
        {
            parametros += "Precisi\u00F3n longitud=";
```

```
            parametros += valor_precision_longitud;
        }
        else if(parametros != "")
        {
            parametros += "&Precisi\u00F3n longitud=";
            parametros += valor_precision_longitud;
        }
    }
    if(precision_altitud_bbdd_local != undefined && precision_altitud_bbdd_local != null)
    {
        valor_precision_altitud = encodeURIComponent(precision_altitud_bbdd_local);
        if(parametros == "")
        {
            parametros += "Precisi\u00F3n altitud=";
            parametros += valor_precision_altitud;
        }
        else if(parametros != "")
        {
            parametros += "&Precisi\u00F3n altitud=";
            parametros += valor_precision_altitud;
        }
    }
    if(velocidad_bbdd_local != undefined && velocidad_bbdd_local != null)
    {
        valor_velocidad = encodeURIComponent(velocidad_bbdd_local);
        if(parametros == "")
        {
            parametros += "Velocidad=";
            parametros += valor_velocidad;
        }
        else if(parametros != "")
        {
            parametros += "&Velocidad=";
            parametros += valor_velocidad;
        }
    }
    if(RPM_motor_bbdd_local != undefined && RPM_motor_bbdd_local != null)
    {
        valor_RPM_motor = encodeURIComponent(RPM_motor_bbdd_local);
        if(parametros == "")
        {
            parametros += "RPM motor=";
            parametros += valor_RPM_motor;
        }
        else if(parametros != "")
        {
            parametros += "&RPM motor=";
            parametros += valor_RPM_motor;
        }
    }
    if(temperatura_motor_bbdd_local != undefined && temperatura_motor_bbdd_local != null)
    {
        valor_temperatura_motor = encodeURIComponent(temperatura_motor_bbdd_local);
        if(parametros == "")
        {
```

```
            parametros += "Temperatura motor=";
            parametros += valor_temperatura_motor;
        }
        else if(parametros != "")
        {
            parametros += "&Temperatura motor=";
            parametros += valor_temperatura_motor;
        }
    }
    if(valor_velocimetro_bbdd_local != undefined && valor_velocimetro_bbdd_local != null)
    {
        valor_valor_velocimetro = encodeURIComponent(valor_velocimetro_bbdd_local);
        if(parametros == "")
        {
            parametros += "Valor veloc\u00EDmetro=";
            parametros += valor_valor_velocimetro;
        }
        else if(parametros != "")
        {
            parametros += "&Valor veloc\u00EDmetro=";
            parametros += valor_valor_velocimetro;
        }
    }
    if(flujo_de_masa_de_aire_bbdd_local != undefined && flujo_de_masa_de_aire_bbdd_local != null)
    {
        valor_flujo_de_masa_de_aire = encodeURIComponent(flujo_de_masa_de_aire_bbdd_local);
        if(parametros == "")
        {
            parametros += "Flujo de masa de aire=";
            parametros += valor_flujo_de_masa_de_aire;
        }
        else if(parametros != "")
        {
            parametros += "&Flujo de masa de aire=";
            parametros += valor_flujo_de_masa_de_aire;
        }
    }
    if(posicion_acelerador_bbdd_local != undefined && posicion_acelerador_bbdd_local != null)
    {
        valor_posicion_acelerador = encodeURIComponent(posicion_acelerador_bbdd_local);
        if(parametros == "")
        {
            parametros += "Posici\u00F3n acelerador=";
            parametros += valor_posicion_acelerador;
        }
        else if(parametros != "")
        {
            parametros += "&Posici\u00F3n acelerador=";
            parametros += valor_posicion_acelerador;
        }
    }
    if(temperatura_ambiente_bbdd_local != undefined && temperatura_ambiente_bbdd_local != null)
    {
        valor_temperatura_ambiente = encodeURIComponent(temperatura_ambiente_bbdd_local);
        if(parametros == "")
        {
```

```
                    parametros += "Temperatura ambiente=";
                    parametros += valor_temperatura_ambiente;
                }
                else if(parametros != "")
                {
                    parametros += "&Temperatura ambiente=";
                    parametros += valor_temperatura_ambiente;
                }
            }
            if(numero_errores_detectados_en_vehiculo_bbdd_local != undefined && numero_errores_detectados_en_vehiculo_bbdd_local != null)
            {
                valor_numero_errores_detectados_en_vehiculo = encodeURIComponent(numero_errores_detectados_en_vehiculo_bbdd_local);
                if(parametros == "")
                {
                    parametros += "N\u00FAmero de errores detectados en el veh\u00EDculo=";
                    parametros += valor_numero_errores_detectados_en_vehiculo;
                }
                else if(parametros != "")
                {
                    parametros += "&N\u00FAmero de errores detectados en el veh\u00EDculo=";
                    parametros += valor_numero_errores_detectados_en_vehiculo;
                }
            }
            if(codigos_errores_detectados_en_vehiculo_bbdd_local != undefined && codigos_errores_detectados_en_vehiculo_bbdd_local !=
null)
            {
                valor_codigos_errores_detectados_en_vehiculo = encodeURIComponent(codigos_errores_detectados_en_vehiculo_bbdd_local);
                if(parametros == "")
                {
                    parametros += "C\u00F3digos de los errores detectados en el veh\u00EDculo=";
                    parametros += valor_codigos_errores_detectados_en_vehiculo;
                }
                else if(parametros != "")
                {
                    parametros += "&C\u00F3digos de los errores detectados en el veh\u00EDculo=";
                    parametros += valor_codigos_errores_detectados_en_vehiculo;
                }
            }
            //Abrimos una nueva petición HTTP e indicamos que es de tipo POST:
            peticion.open("POST", url, true);
            //Indicamos de qué tipo es el contenido de la petición POST:
            peticion.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
            //Enviamos la petición POST pasando como argumento los parámetros correctamente codificados:
            peticion.send(parametros);
        }
        //Una vez que hemos terminado de pasar el contenido de la base de datos almacenada localmente a la base de datos en la nube,
vaciamos la base de datos almacenada localmente (puesto que su contenido ya se ha subido a la base de datos en la nube), indicamos que la
base de datos almacenada localmente ahora se encuentra vacía e indicamos también al resto de la app que ya no se está trabajando en pasar
el contenido de la base de datos almacenada localmente a la base de datos en la nube:
        //alert("Filas de la base de datos antes de 'DELETE FROM PFC': " + resultado.rows.length);
        transaccion_bbdd.executeSql("DELETE FROM PFC");
        //transaccion_bbdd.executeSql("SELECT FROM PFC", [], comprobar_contenido_bbdd_localExito, comprobar_contenido_bbdd_localError);
        base_de_datos_local_vacia = true;
        pasando_bbdd_local_a_bbdd_en_la_nube = false;
    }

    //Función abrir_bbdd_localExito: se la invoca cuando la apertura/creación de la base de datos local tiene éxito.
```

```
    function abrir_bbdd_localExito(bbdd)

    {

        base_de_datos_local_abierta = true;

        //alert("Funcion abrir_bbdd_localExito");

        //Le damos valor a la variable transaccion_bbdd para poder hacer transacciones en la base de datos almacenada localmente en cualquier punto del código de la app:

        transaccion_bbdd = bbdd.beginTransaction();

        //Creamos una única tabla en la base de datos con las columnas correspondientes a los parámetros obtenidos de la interfaz OBD2 del vehículo y a los parámetros obtenidos del terminal Android:

        transaccion_bbdd.executeSql("CREATE TABLE PFC (VIN varchar(20), Fecha varchar(10), Hora varchar(8), Latitud float, Longitud float, Altitud float, Precision_latitud float, Precision_longitud float, Precision_altitud float, Velocidad float, RPM_motor float, Temperatura_motor float, Valor_velocimetro float, Flujo_de_masa_de_aire float, Posicion_acelerador float, Temperatura_ambiente float, Numero_errores_detectados_en_vehiculo float, Codigos_errores_detectados_en_vehiculo varchar(20))");

        //Hacemos que se pueda acceder a la base de datos desde cualquier parte del código de la app:

        base_de_datos_local = bbdd;

    }


    //Función abrir_bbdd_local: función encargada de abrir la base de datos almacenada localmente para ir introduciendo en ella los parámetros que se vayan obteniendo tanto del vehículo como del propio terminal Android.

    function abrir_bbdd_local()

    {

        window.sqlitePlugin.openDatabase({name: 'PFC.db', location: 'default'}, abrir_bbdd_localExito);

    }


    //ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles: función que comprueba cada vez que se la invoca si la ubicación, la comunicación por Bluetooth y la conexión a Internet están disponibles y según el caso programa la invocación cada cierto tiempo de la función que agrega datos bien a la base de datos almacenada localmente o bien a la base de datos almacenada en la nube.

    function ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles()

    {

        //alert("Funcion ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles");

        //Actualizamos el valor de la variable conexion_Internet_disponible por si antes hubiera existido una conexión a Internet y ahora no o viceversa:

        comprueba_conexion_Internet();

        //Si la ubicación y la comunicación por Bluetooth están disponibles y además existe una conexión válida a Internet, entonces programamos cada cierto tiempo la invocación de la función encargada de agregar datos a la base de datos almacenada en la nube:

        if(ubicacion_disponible == true && comunicacion_bluetooth_disponible == true && conexion_Internet_disponible == true)

        {

            //Comprobamos si hay programada de antemano una invocación a la función agregar_fila_bbdd_local, y en caso de que sea así la detenemos:

            if(invocacion_programada_a_funcion_agregar_fila_bbdd_local == true)

            {

                //alert("Caso 1.1");

                clearInterval(agregar_fila_bbdd_local_id);

                invocacion_programada_a_funcion_agregar_fila_bbdd_local = false;

            }

            //En caso de que todavía no haya sido programada de antemano una invocación a la función agregar_fila_bbdd_en_la_nube, entonces programamos la invocación cada cierto tiempo de la función agregar_fila_bbdd_en_la_nube:

            if(invocacion_programada_a_funcion_agregar_fila_bbdd_en_la_nube == false)

            {

                //alert("Caso 1.2");

                agregar_fila_bbdd_en_la_nube_id = setInterval(agregar_fila_bbdd_en_la_nube, periodo_almacenamiento_bbdd);

                invocacion_programada_a_funcion_agregar_fila_bbdd_en_la_nube = true;

            }

            //Si la base de datos almacenada localmente está abierta, hay registros en ella y no se está actualmente trabajando en el traspaso del contenido de la base de datos almacenada localmente a la base de datos en la nube, intentamos pasar el contenido de la base de datos almacenada localmente a la base de datos en la nube:

            if(base_de_datos_local != null && base_de_datos_local != undefined && transaccion_bbdd != null && transaccion_bbdd != undefined && base_de_datos_local_vacia == false && pasando_bbdd_local_a_bbdd_en_la_nube == false)

            {

                //alert("Caso 1.3");

                transaccion_bbdd.executeSql("SELECT * FROM PFC", [], pasar_contenido_bbdd_local_a_bbdd_en_la_nubeExito, pasar_contenido_bbdd_local_a_bbdd_en_la_nubeError);

            }
```

```
        }

        //En cambio, si la ubicación y la comunicación por Bluetooth están disponibles pero no existe una conexión válida a Internet, entonces programamos cada cierto tiempo la invocación de la función encargada de agregar datos a la base de datos almacenada localmente en el propio terminal Android:

        else if(ubicacion_disponible == true && comunicacion_bluetooth_disponible == true && conexion_Internet_disponible == false)

        {

            //Comprobamos si la base de datos almacenada localmente todavía no se ha abierto, y en ese caso la abrimos:

            if(base_de_datos_local_abierta == false)

            {

                //alert("Caso 2.1");

                abrir_bbdd_local();

            }

            //Comprobamos si hay programada de antemano una invocación a la función agregar_fila_bbdd_en_la_nube, y en caso de que sea así la detenemos.

            if(invocacion_programada_a_funcion_agregar_fila_bbdd_en_la_nube == true)

            {

                //alert("Caso 2.2");

                clearInterval(agregar_fila_bbdd_en_la_nube_id);

                invocacion_programada_a_funcion_agregar_fila_bbdd_en_la_nube = false;

            }

            //En caso de que todavía no haya sido programada de antemano una invocación a la función agregar_fila_bbdd_local (o de que esta se haya "limpiado"), entonces programamos la invocación cada cierto tiempo de la función agregar_fila_bbdd_local:

            if(invocacion_programada_a_funcion_agregar_fila_bbdd_local == false)

            {

                //alert("Caso 2.3");

                agregar_fila_bbdd_local_id = setInterval(agregar_fila_bbdd_local, periodo_almacenamiento_bbdd);

                invocacion_programada_a_funcion_agregar_fila_bbdd_local = true;

            }

        }

    }


    //Función desconexion_bluetoothExito: se la invoca cuando la desconexión del adaptador Bluetooth tiene exito.

    function desconexion_bluetoothExito()

    {

        //alert("Funcion desconexion_bluetoothExito.");

        //Informamos al usuario a través de la tabla relativa a los parámetros obtenidos del terminal del cambio producido en la conexión por Bluetooth del terminal con el vehículo:

        document.getElementById("estado_conexion_bluetooth").innerHTML = "<p align='center'>Esperando que el usuario valide la configuraci&oacute;n...</p>";

        //Limpiamos la invocación programada a la función comunicacion_bluetooth_interrumpida, ya que la invocaremos manualmente para que no haya que esperar aleatoriamente entre 0 y 5 segundos para indicarle al usuario que la comunicación por Bluetooth con el vehículo ha quedado interrumpida:

        clearTimeout(comunicacion_bluetooth_interrumpida_id);

        comunicacion_bluetooth_interrumpida();

        //Nos desubscribimos a la recepción de datos por Bluetooth:

        bluetoothSerial.unsubscribeRawData();

        //Indicamos a otras partes del código de la app que la comunicación por Bluetooth con el vehículo ya no está disponible.

        comunicacion_bluetooth_disponible = false;

        //Limpiamos la invocación cada cierto tiempo a la función gestionar_comunicacion_bluetooth:

        clearInterval(gestionar_comunicacion_bluetooth_id);

        //alert("Valor esperando_recepcion_por_bluetooth: " + esperando_recepcion_por_bluetooth);

    }


    //Función desconexion_bluetoothError: se la invoca cuando la desconexión del adaptador Bluetooth resulta en error.

    function desconexion_bluetoothError()

    {

        //alert("Error al realizar la desconexión del adaptador Bluetooth.");

    }
```

```
    //Función borrar_valores_parametros_obtenidos_interfaz_OBD2_vehiculo: función encargada de borrar las filas de la columna "Valor" de la tabla relativa a los parámetros obtenidos de la interfaz OBD2 del vehículo excepto las filas correspondientes a los parámetros número de identificación del vehículo (VIN), número de errores detectados en el vehículo y códigos de error de los errores detectados en el vehículo.

    function borrar_valores_parametros_obtenidos_interfaz_OBD2_vehiculo()

    {

        document.getElementById("RPM_motor").innerHTML = "";

        document.getElementById("posicion_acelerador").innerHTML = "";

        document.getElementById("temperatura_motor").innerHTML = "";

        document.getElementById("flujo_de_masa_de_aire").innerHTML = "";

        document.getElementById("valor_velocimetro").innerHTML = "";

        document.getElementById("temperatura_ambiente").innerHTML = "";

    }


    //Función comunicacion_Internet_interrumpida: cambia el led verde parpadeante a rojo fijo para indicar que se ha interrumpido la conexión establecida que había con la base de datos en la nube.

    function comunicacion_Internet_interrumpida()

    {

        document.getElementById("estado_comunicacion_Internet").innerHTML = "<p align='center'><img src='img/led_rojo.png'><br>Comunicaci&oacute;n interrumpida.</p>";

    }


    //Función comprobar_validacion_configuracion: comprueba cada vez que se la invoca si se ha validado la configuración de los parámetros relativos a la conexión con la base de datos almacenada en la nube y de la comunicación por Bluetooth con el vehículo.

    function comprobar_validacion_o_detencion_configuracion()

    {

        if(configuracion_validada == true)

        {

            configuracion_interrumpida = false;

            //Guardamos en las variables de configuración los valores de los parámetros de configuración correspondientes introducidos en la tabla de configuración.

            periodo_almacenamiento_bbdd = datos_configuracion.periodo_almacenamiento_bbdd.value;

            //Guardamos en el almacenamiento local el periodo de almacenamiento en la base de datos para que se recuerde en la siguiente ejecución de la app:

            window.localStorage.setItem('periodo_almacenamiento_bbdd', periodo_almacenamiento_bbdd);

            dispositivo_bluetooth_seleccionado = datos_configuracion.dispositivo_bluetooth.value;

            email_cuenta_Zoho = datos_configuracion.email_cuenta_Zoho.value;

            //Guardamos en el almacenamiento local el email de la cuenta Zoho para que se recuerde en la siguiente ejecución de la app:

            window.localStorage.setItem('email_cuenta_Zoho', email_cuenta_Zoho);

            authtoken = datos_configuracion.authtoken.value;

            //Guardamos en el almacenamiento local el token de autenticación para que se recuerde en la siguiente ejecución de la app:

            window.localStorage.setItem('authtoken', authtoken);

            nombre_bbdd = datos_configuracion.nombre_bbdd.value;

            //Guardamos en el almacenamiento local el nombre de la base de datos para que se recuerde en la siguiente ejecución de la app:

            window.localStorage.setItem('nombre_bbdd', nombre_bbdd);

            nombre_tabla_bbdd = datos_configuracion.nombre_tabla_bbdd.value;

            //Guardamos en el almacenamiento local el nombre de la tabla de la base de datos para que se recuerde en la siguiente ejecución de la app:

            window.localStorage.setItem('nombre_tabla_bbdd', nombre_tabla_bbdd);

            MAC_adaptador_bluetooth = dispositivos_bluetooth_emparejados[dispositivo_bluetooth_seleccionado].address;

            //alert("Parametros configuracion:\n" + periodo_almacenamiento_bbdd + "\n" + dispositivo_bluetooth_seleccionado + "\n" + email_cuenta_Zoho + "\n" + authtoken + "\n" + nombre_bbdd + "\n" + nombre_tabla_bbdd + "\n" + MAC_adaptador_bluetooth);

            //Nos conectamos con la ECU (Engine Control Unit) del vehículo a través del adaptador Bluetooth seleccionado:

            bluetoothSerial.connect(MAC_adaptador_bluetooth, conexion_bluetoothExito, conexion_bluetoothError);

            //Comprobamos una vez cada tres segundos si tanto la ubicación como la comunicación por Bluetooth y la conexión a Internet están disponibles para empezar a agregar datos a la base de datos almacenada localmente o en la nube:

            ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles_id = setInterval(ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles, tres_segundos);

            //Hacemos que los valores de la tabla de configuración ya no se puedan cambiar (parámetro disabled = true):

            formulario_configuracion = document.getElementsByName("formulario_configuracion")[0];

            //alert("Número de elementos del formulario de configuracion: " + formulario_configuracion.elements.length);
```

```
            for(i=0; i<formulario_configuracion.elements.length; i++)
            {
                //Deshabilitamos todos los elementos del formulario salvo el número 7: el botón "DETENER", que debe quedar habilitado:
                if(i != numero_elemento_boton_detener)
                {
                   formulario_configuracion.elements[i].disabled = true;
                }
                else if(i == numero_elemento_boton_detener)
                {
                    formulario_configuracion.elements[i].disabled = false;
                }
            }
            //Si existe conexión a Internet al pulsar el boton "VALIDAR", programamos la invocación de la función comunicacion_Internet_interrumpida justo cuando transcurran dos veces el periodo de almacenamiento de datos en la base de datos en la nube, tiempo suficiente para que se cree al menos una nueva petición POST que limpie la invocación de esta función antes de que esta se lleve a cabo:
            if(conexion_Internet_disponible == true)
            {
                comunicacion_Internet_interrumpida_id = setTimeout(comunicacion_Internet_interrumpida, periodo_almacenamiento_bbdd*2);
            }
            //Cambiamos el valor de la bandera para que todo lo que se tiene que ejecutar al validar la configuración solamente se ejecute una única vez:
            configuracion_validada = false;
        }
        if(configuracion_detenida == true)
        {
            configuracion_interrumpida = true;
            //alert("Configuracion detenida");
            //Detenemos la invocación programada cada 3 segundos a la función que comprueba si está disponible la ubicación, la comunicación por Bluetooth y la conexión a Internet:
            clearInterval(ubicacion_comunicacion_bluetooth_y_conexion_a_Internet_disponibles_id);
            //Nos desconectamos del adaptador Bluetooth:
            bluetoothSerial.disconnect(desconexion_bluetoothExito, desconexion_bluetoothError);
            //En caso de que exista una invocación programada a la función agregar_fila_bbdd_local la limpiamos:
            if(invocacion_programada_a_funcion_agregar_fila_bbdd_local == true)
            {
                clearInterval(agregar_fila_bbdd_local_id);
                invocacion_programada_a_funcion_agregar_fila_bbdd_local = false;
            }
            //En caso de que exista una invocación programada a la función agregar_fila_bbdd_en_la_nube la limpiamos:
            if(invocacion_programada_a_funcion_agregar_fila_bbdd_en_la_nube == true)
            {
                clearInterval(agregar_fila_bbdd_en_la_nube_id);
                invocacion_programada_a_funcion_agregar_fila_bbdd_en_la_nube = false;
            }
            //Hacemos que los parámetros de la tabla de configuración se puedan volver a editar:
            formulario_configuracion = document.getElementsByName("formulario_configuracion")[0];
            for(i=0; i<formulario_configuracion.elements.length; i++)
            {
                //Habilitamos todos los elementos del formulario salvo el número 7: el botón "DETENER", que debe quedar deshabilitado:
                if(i != numero_elemento_boton_detener)
                {
                    formulario_configuracion.elements[i].disabled = false;
                }
                else if(i == numero_elemento_boton_detener)
                {
                    formulario_configuracion.elements[i].disabled = true;
```

```
                }

            }

            //Borramos las filas de la columna "Valor" de la tabla de los parámetros obtenidos de la interfaz OBD2 del vehículo escepto las filas correspondientes a los parámetros número de identificación del vehículo (VIN), número de errores detectados en el vehículo y códigos de error de los errores detectados en el vehículo, ya que no tiene sentido mantener valores obsoletos de parámetros obtenidos de la interfaz OBD2 del vehículo una vez que la comunicación por Bluetooth ha sido detenida.

            setTimeout(borrar_valores_parametros_obtenidos_interfaz_OBD2_vehiculo, time_out_detencion_configuracion);

            //Si al pulsar el botón "DETENER" existe una conexión a Internet Dado que se va a detener la comunicación con la base de datos en la nube, se invoca a la función comunicacion_Internet_interrumpida para cambiar el led verde parpadeante a rojo y se limpia al mismo tiempo la invocación temporizada a esta función que se hizo con anterioridad al pulsar el botón "VALIDAR":

            if(conexion_Internet_disponible == true)

            {

                comunicacion_Internet_interrumpida();

                clearTimeout(comunicacion_Internet_interrumpida_id);

            }

            //Cambiamos el valor de la bandera de entrada al cuerpo del if para que este solamente se ejecute una vez.

            configuracion_detenida = false;

        }

    }


    //Función ubicacion_habilitadaExito: se la invoca en caso de que el acceso a la información del terminal sobre la habilitación o la deshabilitación de la ubicación o localización de este tenga éxito:

    function ubicacion_habilitadaExito(ubicacion)

    {

        if(ubicacion == true)

        {

            ubicacion_habilitada = true;

        }

        else

        {

            ubicacion_habilitada = false;

        }

    }


    //Función ubicacion_habilitadaError: se la invoca en caso de que el acceso a la información del terminal sobre la habilitación o deshabilitación de la ubicación o localización de este resulte en error:

    function ubicacion_habilitadaError(error)

    {

        alert("Funci\u00F3n ubicacion_habilitadaError");

    }


    //Función bluetooth_habilitadoExito: se la invocará en caso de que la interfaz Bluetooth del terminal se encuentre habilitada.

    function bluetooth_si_habilitado()

    {

        bluetooth_habilitado = true;

    }


    //Función bluetooth_habilitadoError: se la invocará en caso de que el la interfaz Bluetooh del terminal se encuentre deshabilitada.

    function bluetooth_no_habilitado()

    {

        bluetooth_habilitado = false;

    }


    //Función comprobar_ubicacion_y_bluetooth_habilitados: se la invoca cada 250ms con el objetivo de determinar si el terminal tiene habilitada la interfaz Bluetooth así como el acceso a su ubicación, aunque lo más normal es que solamente sea necesaria una única invocación al principio.

    function comprobar_ubicacion_y_bluetooth_habilitados()

    {

        if(ubicacion_habilitada != undefined && bluetooth_habilitado != undefined)

        {
```

```
            clearInterval(comprobar_ubicacion_y_bluetooth_habilitados_id);

            if(ubicacion_habilitada == true && bluetooth_habilitado == true)

            {

                //Listamos los dispositivos Bluetooth emparejados con el terminal Android con el objetivo de que el usuario de la app elija uno de ellos:

                bluetoothSerial.list(listar_dispositivos_BluetoothExito);

                //Obtenemos la ubicación del terminal:

                navigator.geolocation.getCurrentPosition(obtener_ubicacionExito, obtener_ubicacionError, {enableHighAccuracy: true});

                //Hacemos que se actualicen los parámetros o las coordenadas de la ubicación del terminal cuando este cambie físicamente de ubicación.

                watchLocationId     =     navigator.geolocation.watchPosition(actualizar_ubicacionExito,     actualizar_ubicacionError, {enableHighAccuracy: true});

                document.getElementById("estado_obtencion_ubicacion").innerHTML           =           "<p           align='center'><img src='img/icono_GPS_animado.gif'> <br> Obteniendo la ubicaci&oacute;n del terminal.</p>";

            }

            else if(ubicacion_habilitada == false || bluetooth_habilitado == false)

            {

                if(ubicacion_habilitada == false && bluetooth_habilitado == false)

                {

                    navigator.notification.alert("Se ha detectado que el acceso a la ubicaci\u00F3n o localizaci\u00F3n del terminal as\u00ED como su interfaz Bluetooth no est\u00E1n habilitados. Sin embargo, para que la app pueda llevar a cabo su cometido correctamente es necesario que en la configuraci\u00F3n del terminal tanto el acceso a su ubicaci\u00F3n o localizaci\u00F3n como su interfaz Bluetooth est\u00E9n habilitados. Pulse 'Aceptar' para cerrar la app. Posteriormente, habilite el acceso a la ubicaci\u00F3n o localizaci\u00F3n del terminal y su interfaz Bluetooth. Por \u00FAltimo, vuelva a iniciar la app.", cerrarApp, "Ubicaci\u00F3n y Bluetooth no habilitados", "Aceptar");

                }

                else

                {

                    //alert("Caso 2");

                    //En caso de que solamente esté deshabilitado el acceso a la ubicación o localización del terminal, mostramos un mensaje de advertencia y cerramos la app:

                    if(ubicacion_habilitada == false)

                    {

                        navigator.notification.alert("Se ha detectado que el acceso a la ubicaci\u00F3n o localizaci\u00F3n del terminal no est\u00E1 habilitado. Sin embargo, para que la app pueda llevar a cabo su cometido correctamente es necesario que en la configuraci\u00F3n del terminal el acceso a su ubicaci\u00F3n o localizaci\u00F3n est\u00E9 habilitado. Pulse 'Aceptar' para cerrar la app. Posteriormente, habilite el acceso a la ubicaci\u00F3n o localizaci\u00F3n del terminal y vuelva a iniciar la app.", cerrarApp, "Ubicaci\u00F3n no habilitada", "Aceptar");

                    }

                    //En caso de que solamente esté deshabilitada la interfaz Bluetooth del terminal, mostramos un mensaje de advertencia y cerramos la app:

                    if(bluetooth_habilitado == false)

                    {

                        navigator.notification.alert("Se ha detectado que la interfaz Bluetooth del terminal no est\u00E1 habilitada. Sin embargo, para que la app pueda llevar a cabo su cometido correctamente es necesario que en la configuraci\u00F3n del terminal su interfaz Bluetooth est\u00E9 habilitada. Pulse 'Aceptar' para cerrar la app. Posteriormente, habilite la interfaz Bluetooth del terminal y vuelva a iniciar la app.", cerrarApp, "Bluetooth no habilitado", "Aceptar");

                    }

                }

            }

        }

    }


    //Función gestionar_mensaje_aviso_sin_conexion_Internet: decide qué hacer según qué botón se pulsó cuando aparecio el mensaje que avisaba de que el terminal no tenía disponible una conexión a Internet válida.

    function gestionar_mensaje_aviso_sin_conexion_Internet(indice)

    {

        //Si se pulsó el botón "Cancelar" o se cerró el mensaje, cerramos la app:

        if(indice == 0 || indice == 2)

        {

            //alert("Se pulso Cancelar o se cerro la ventana.");

            cerrarApp();

        }

        //Pero si se pulsó el botón "Aceptar", entonces abrimos la base de datos local para continuar en modo offline:
```

```
        else if(indice == 1)

        {

            //alert("Se pulso Aceptar");

            abrir_bbdd_local();

        }

    }


    //Lo primero que se hace es comprobar si Bluetooth y el acceso a la ubicación o localización del terminal están habilitados para posteriormente mostrar un mensaje de advertencia en caso de que esto no sea así:

    cordova.plugins.diagnostic.isLocationEnabled(ubicacion_habilitadaExito, ubicacion_habilitadaError);

    bluetoothSerial.isEnabled(bluetooth_si_habilitado, bluetooth_no_habilitado);

    //Hacemos que si la app pasa a un segundo plano siga ejecutándose igual que si estuviera en primer plano:

    cordova.plugins.backgroundMode.enable();

    cordova.plugins.backgroundMode.on('activate', function(){cordova.plugins.backgroundMode.disableWebViewOptimizations();});

    //Hacemos que si se pulsa el botón de ir hacia atrás la app no se cierre, sino que pase a un segundo plano:

    cordova.plugins.backgroundMode.overrideBackButton();

    //Programamos la invocación una vez cada medio segundo de la función comprobar_validacion_configuracion, que comprobará si se ha pulsado el botón "VALIDAR" o el botón "DETENER":

    comprobar_validacion_o_detencion_configuracion_id = setInterval(comprobar_validacion_o_detencion_configuracion, medio_segundo);

    //Hacemos que el formulario de la tabla de los parámetros de configuración recuerde los valores de los parámetros de configuración de la última ejecución:

    document.getElementsByName("periodo_almacenamiento_bbdd")[0].value = window.localStorage.getItem("periodo_almacenamiento_bbdd");

    document.getElementsByName("email_cuenta_Zoho")[0].value = window.localStorage.getItem("email_cuenta_Zoho");

    document.getElementsByName("authtoken")[0].value = window.localStorage.getItem("authtoken");

    document.getElementsByName("nombre_bbdd")[0].value = window.localStorage.getItem("nombre_bbdd");

    document.getElementsByName("nombre_tabla_bbdd")[0].value = window.localStorage.getItem("nombre_tabla_bbdd");

    //Añadimos o agregamos los botones "VALIDAR" y "DETENER" una vez que el terminal está listo, antes no:

    document.getElementById("botones_validar_y_detener").innerHTML  +=  "<p  align='center'><input  id='boton_gordo'  name='boton_validar' type='submit' value='VALIDAR'></p>";

    document.getElementById("botones_validar_y_detener").innerHTML  +=  "<p  align='center'><input  id='boton_gordo'  name='boton_detener' type='button' onclick='detener_configuracion()' value='DETENER' disabled></p>";

    //Comprobamos si existe una conexión a Internet válida en el terminal:

    comprueba_conexion_Internet();

    if(conexion_Internet_disponible == false)

    {

        navigator.notification.confirm("No se ha detectado ninguna conexi\u00F3n v\u00E1lida a Internet. Es recomendable usar esta app con conexi\u00F3n a Internet, dado que si no no se podr\u00E1n enviar los datos recopilados a la base de datos en la nube. De todas formas, tambi\u00E9n es posible usar la app en modo offline. Por lo tanto, para continuar con la ejecuci\u00F3n de la app en modo offline pulse 'Aceptar'. De lo contrario, pulse 'Cancelar' para cerrar la app y poder volver a iniciarla una vez que haya habilitado una conexi\u00F3n v\u00E1lida a Internet en el terminal.", gestionar_mensaje_aviso_sin_conexion_Internet, "Sin conexi\u00F3n a Internet", ["Aceptar", "Cancelar"]);

    }

    //Comprobamos cada medio segundo si se ha determinado el estado de activación o desactivación del acceso a la ubicación o localización del terminal así como de su interfaz Bluetooth. Normalmente bastará con la primera llamada una vez transcurridos 0,25 segundos, pero al tratarse de una ejecución asíncrona por si acaso hacemos más llamadas hasta que se tenga acceso a esta información.

    comprobar_ubicacion_y_bluetooth_habilitados_id = setInterval(comprobar_ubicacion_y_bluetooth_habilitados, un_cuarto_de_segundo);

}
```

ANEXO B: INSTRUCCIONES DE COMPILACIÓN

Para compilar la app del proyecto, como es lógico, es necesario disponer de su código fuente. Este, además de haberse transcrito en formato texto en el anexo A, también está disponible en forma de archivo comprimido con extensión *rar*. Si desea obtener una copia del mismo, puede ponerse en contacto con el autor del proyecto en la dirección de correo electrónico pfc.caja.negra@gmail.com.

Supongamos que el lector ya dispone de una copia del archivo comprimido que contiene el código fuente mencionado anteriormente. Para compilar el proyecto, en primer lugar, habría que descomprimir el archivo comprimido con extensión *rar* en cualquier directorio directorio de una unidad de almacenamiento interno o externo conectada al ordenador del usuario. El contenido de este archivo *rar* es un único directorio de nombre *PFC* que contiene toda la estructura de ficheros y directorios del proyecto, incluidos los plugins.

Una vez que se ha descomprimido el archivo *rar* (y suponiendo que ya se ha llevado a cabo previamente la instalación de entorno de desarrollo Intel XDK) se trataría de ejecutar este software e indicarle que se desea abrir un nuevo proyecto ya existente. Para ello, habría que hacer clic en la esquina inferior izquierda en el botón *OPEN AN INTEL XDK PROJECT*, indicado con el icono de una especie de flecha blanca sobre fondo verde (ver figura 19). Comentar que las instrucciones de compilación que se están dando son para el caso de la versión en lengua inglesa del entorno de desarrollo Intel XDK. Para versiones en otros idiomas (como por ejemplo español), habría que seguir menús o indicaciones análogas.

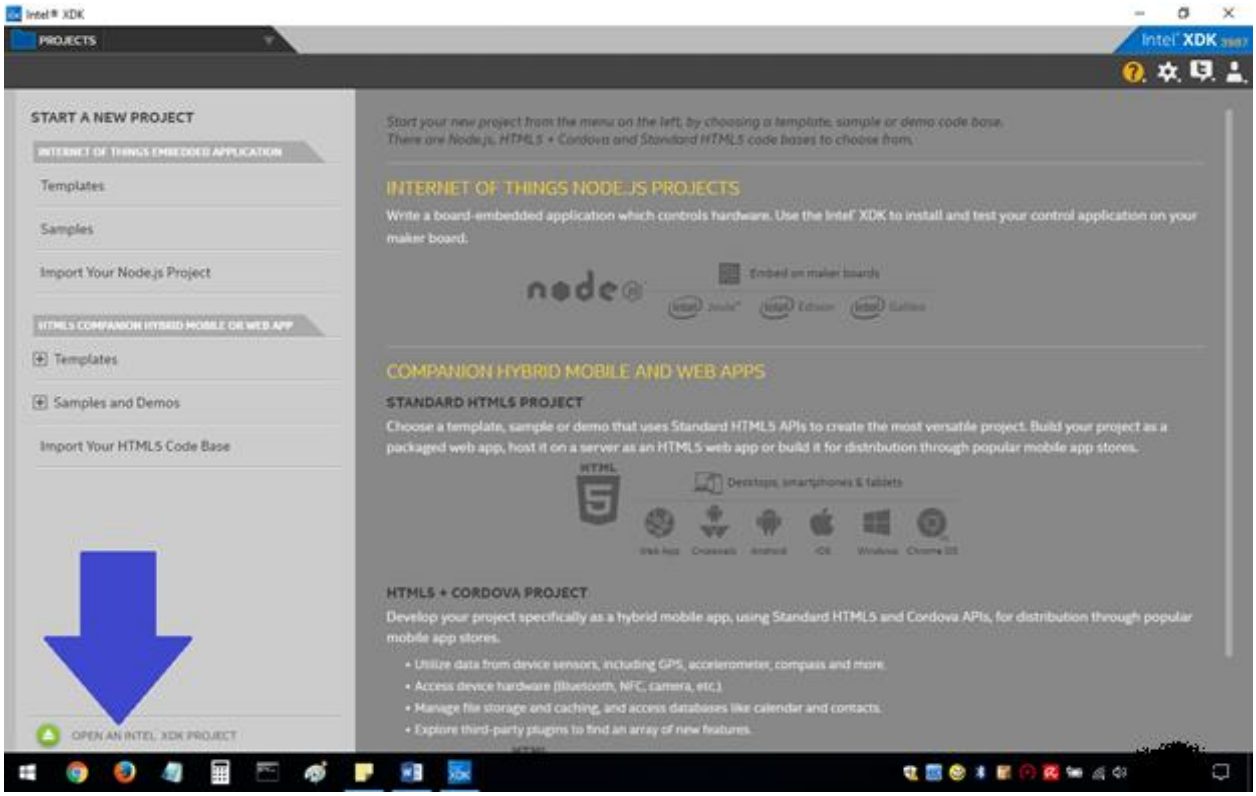

Figura 27: pasos para importar un proyecto en el entorno de desarrollo Intel XDK.

Una vez hecho esto, se nos abrirá una ventana en la que deberemos seleccionar el archivo con nombre *PFC.xdk* localizado en el interior del directorio *PFC* extraido anteriormente del archivo comprimido con extensión *rar*.

El siguiente paso, una vez que se ha importado el proyecto, es ir a la pestaña *BUILD* (ver figura 28), y una vez dentro de ella, deseleccionar, en caso de que estuvieran seleccionados, los sistemas operativos objetivo Windows Phone y iOS haciendo clic en sus respectivos iconos (ver pasos 1 y 2 de la figura 29), dado que la app se ha programado inicialmente solo para el sistema operativo Android. También sería posible compilarla para los sistemas operativos Windows Phone y iOS, pero habría que hacer ciertas modificaciones y adaptaciones en el código fuente de la app que quedan fuera del alcance de este proyecto fin de carrera.

Una vez que los sistemas operativos objetivo Windows Phone y iOS hayan quedado deseleccionados (o ya lo hubieran estado desde un principio), el aspecto que debe tener el contenido de la pestaña BUILD debe ser similar al mostrado en la figura 30.

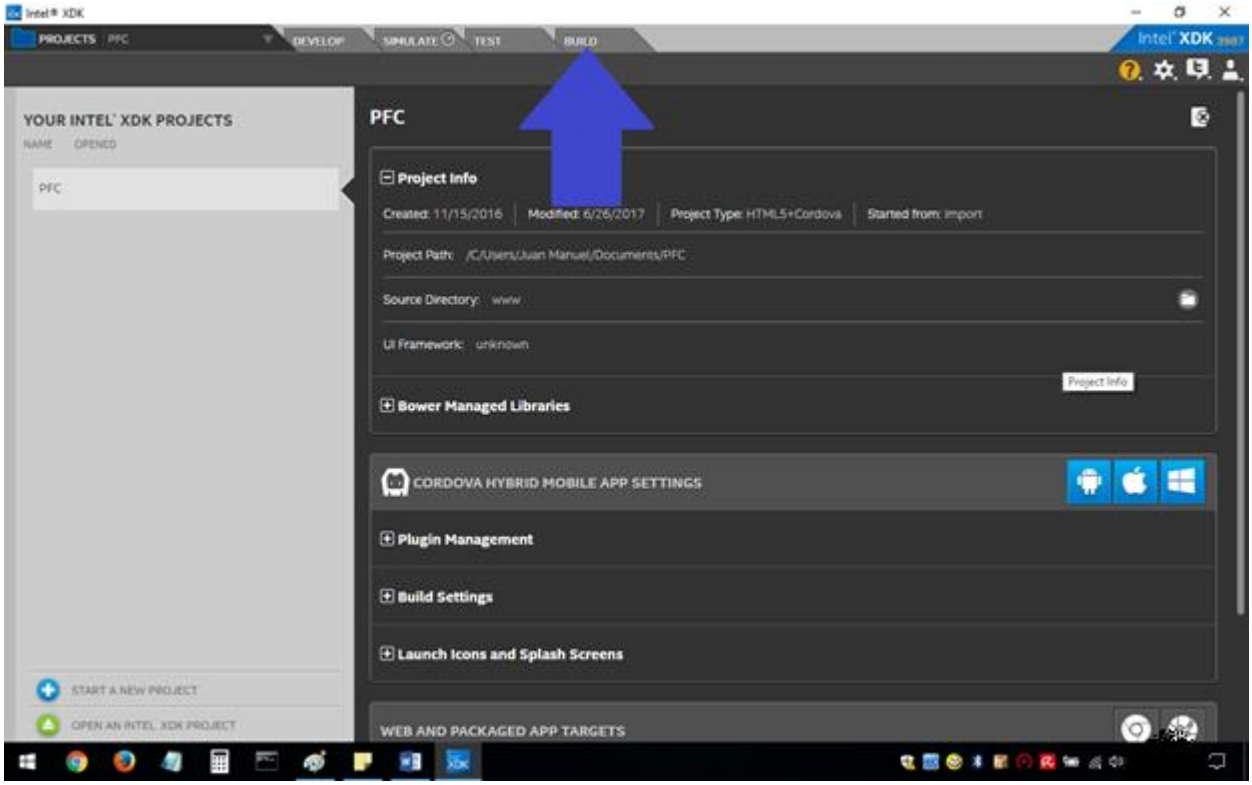

Figura 28: ubicación de la pestaña *BUILD* en el entorno de desarrollo Intel XDK una vez que el proyecto ha sido abierto.

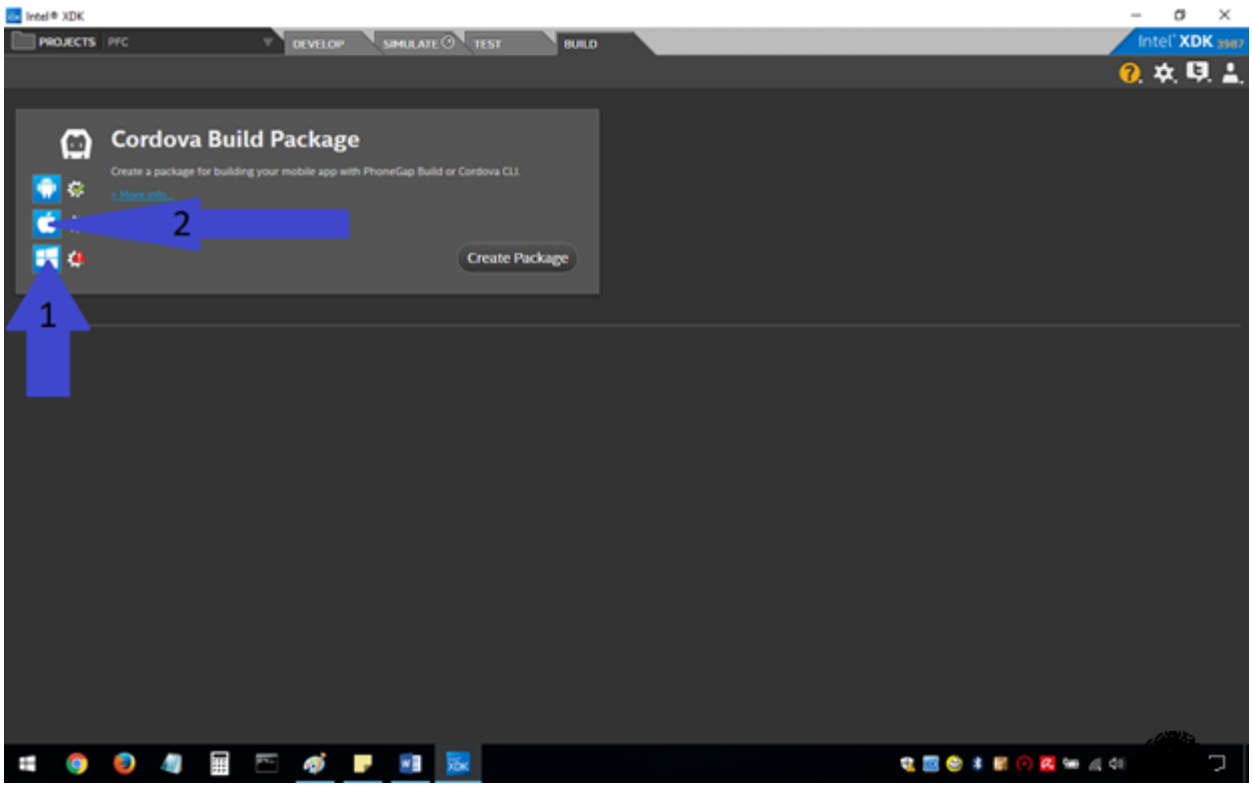

Figura 29: ubicación de los botones de selección/deselección de los sistemas operativos objetivo.

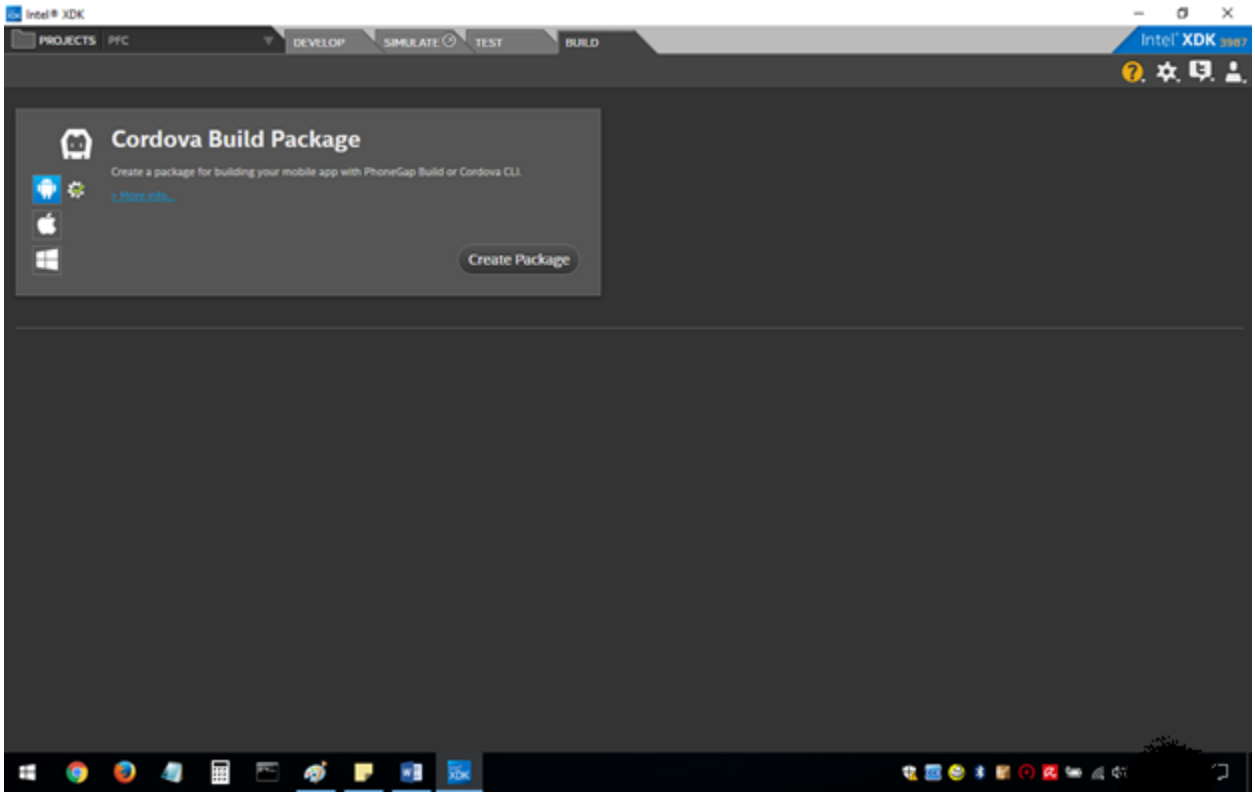

Figura 30: contenido de la pestaña *BUILD* una vez deseleccionados los sistemas operativos objetivo Windows Phone y iOS.

A continuación, habría que hacer clic en el botón *Create Package* (ver figura 31), que abrirá una ventana en la que debemos seleccionar el directorio destino en el que se guardará un archivo comprimido que necesitaremos para compilar la app usando el servicio *Adobe PhoneGap Build*.

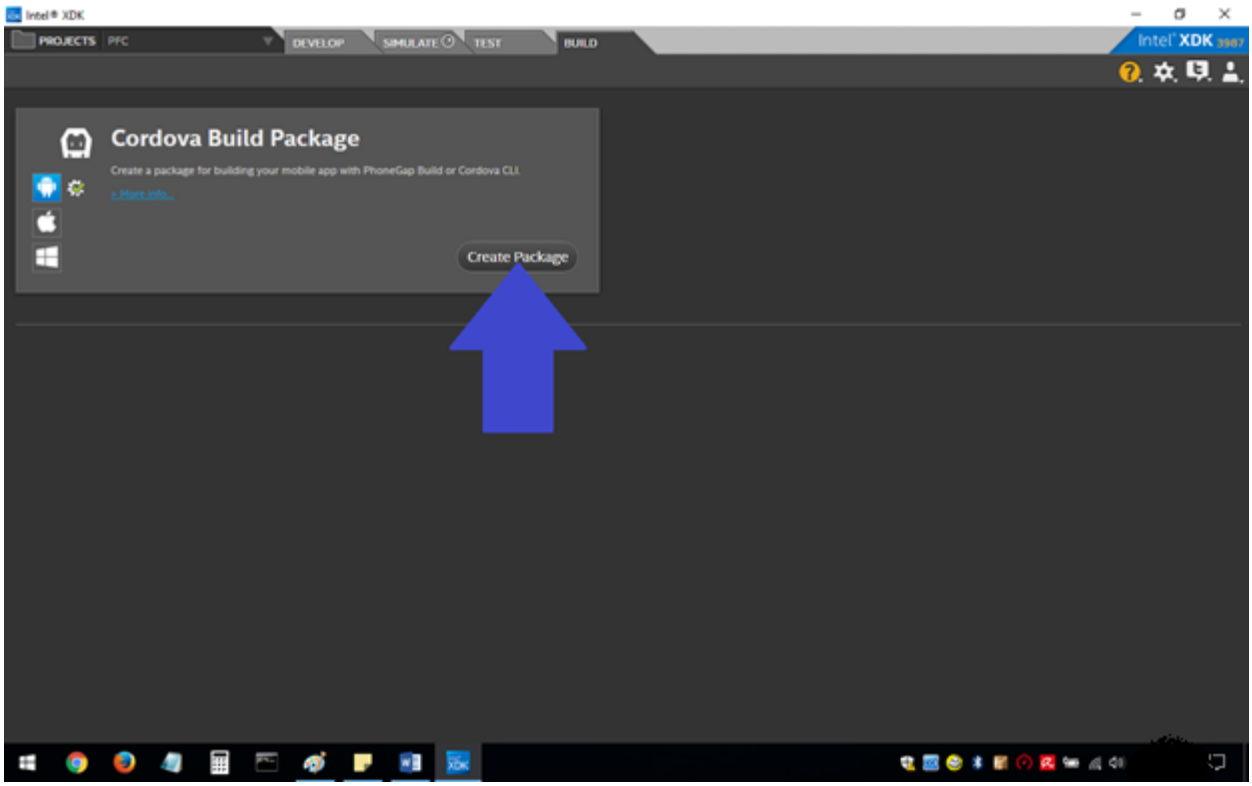

Figura 31: ubicación del botón *Create Package* dentro de la pestaña *BUILD*.

Para compilar la app usando el servicio *Adobe PhoneGap Build* debemos en primer lugar entrar al siguiente enlace: https://build.phonegap.com/. Una vez cargada la web, hariamos clic en el enlace *Sign in* situado arriba a la derecha (ver figura 32) con la intención de iniciar sesión en el servicio.

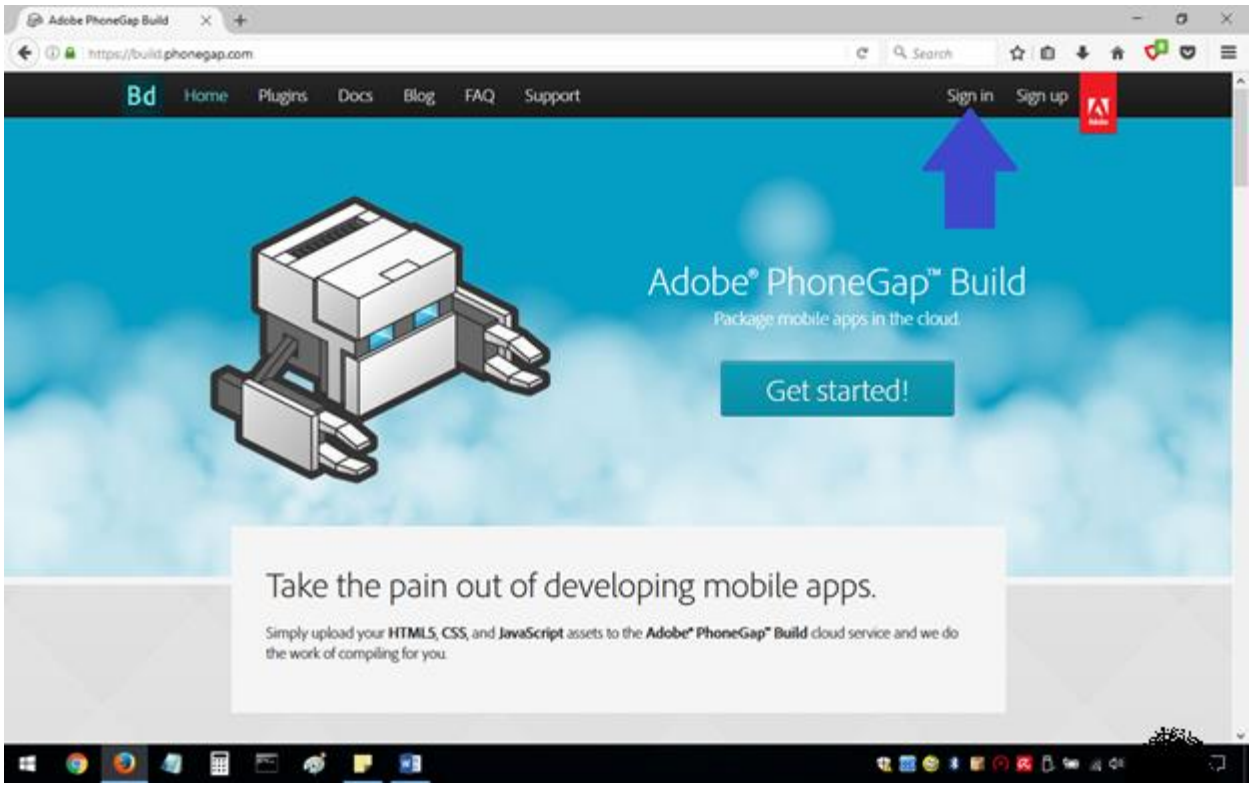

Figura 32: ubicación del enlace *Sign in* del servicio *Adobe PhoneGap Build*.

Se nos abrirá la web mostrada en la figura 33, en la que tendríamos que hacer clic donde dice *Sign up for a new account*.

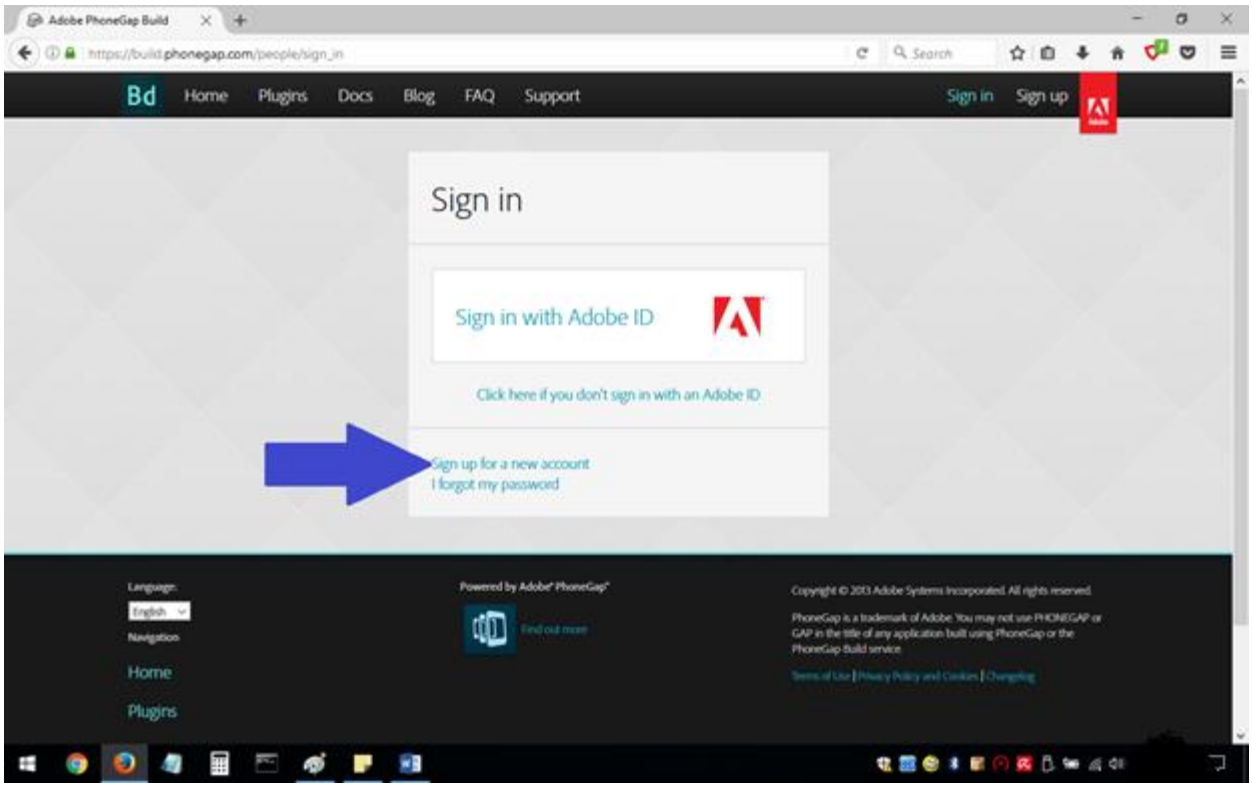

Figura 33: ubicación del enlace *Sign up for a new account*.

Se nos abrirá a continuación la web de la figura 34, en la que tendremos que hacer clic en el botón que dice *completely free*, puesto que el resto de opciones son de pago y se pueden contratar si más adelante hicieran falta.

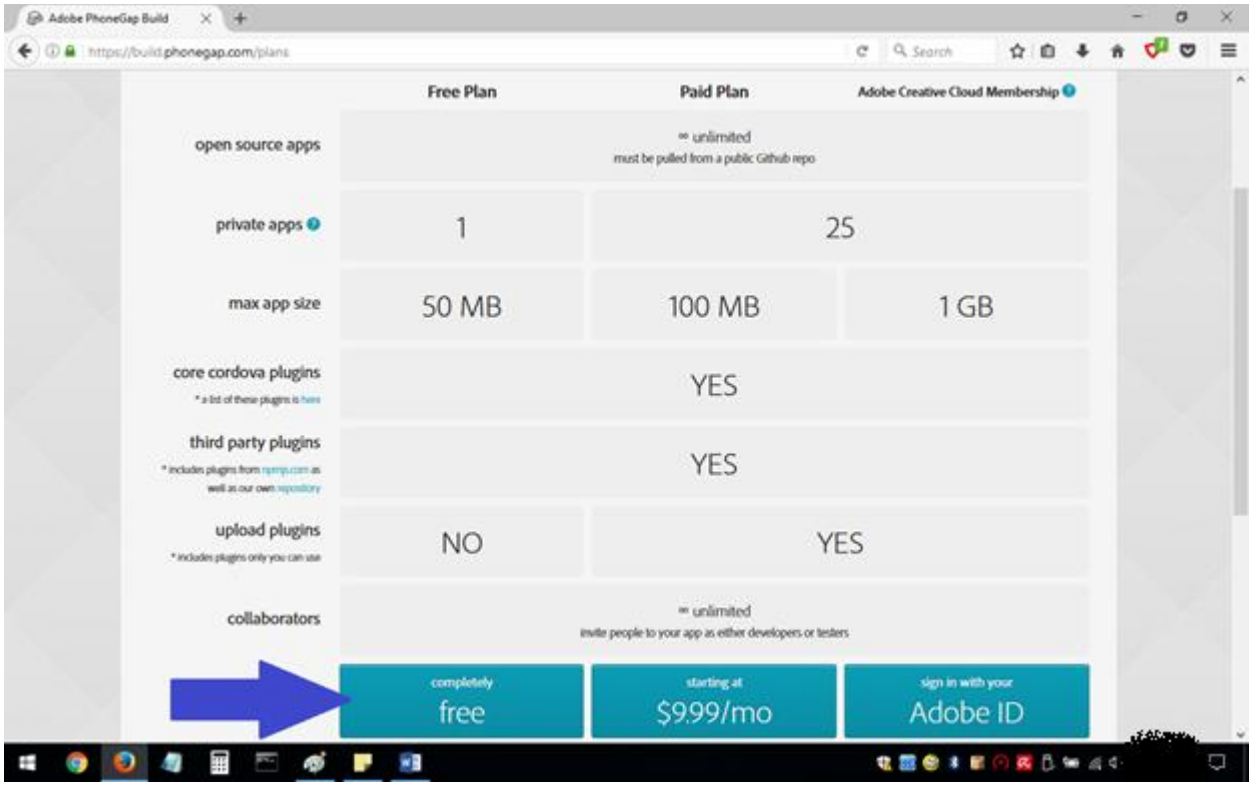

Figura 34: ubicación del botón *completely free*.

Se abrirá la web de la figura 35, en la que debemos hacer clic en el enlace que dice *Get an Adobe ID*.

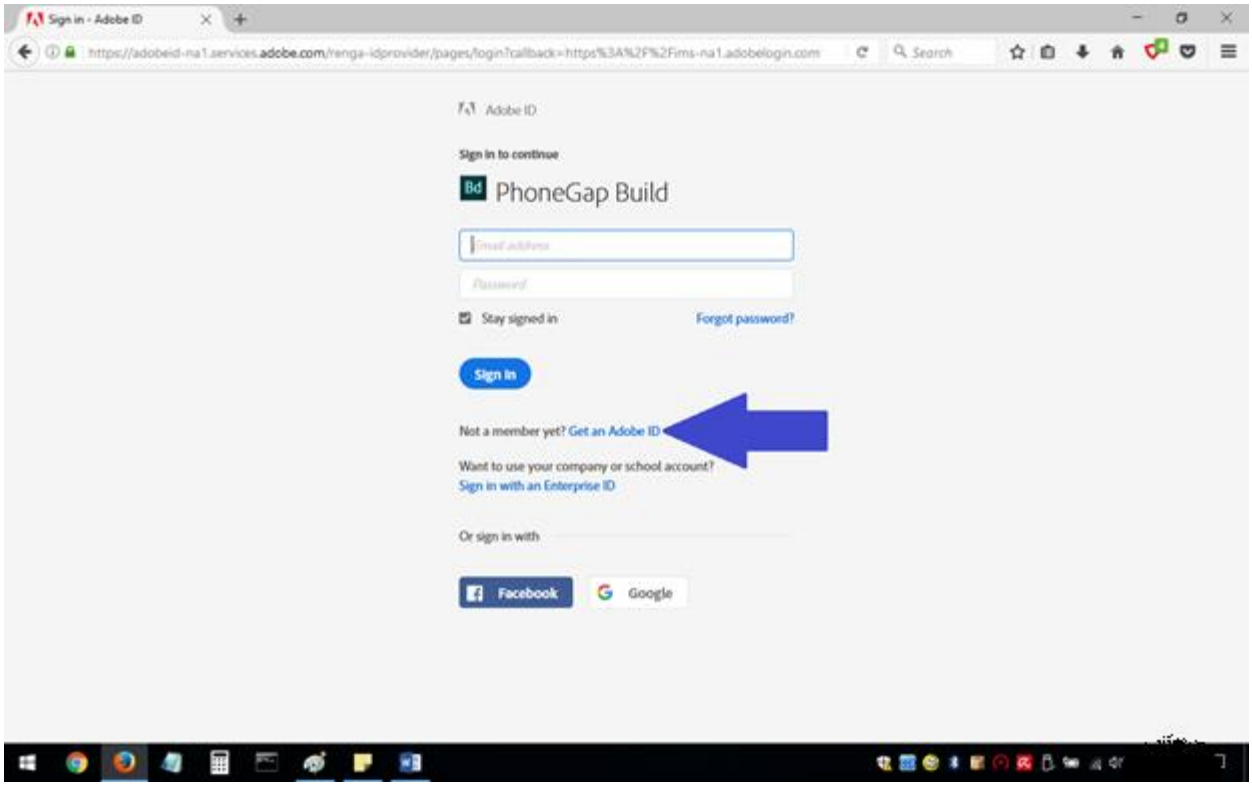

Figura 35: ubicación del enlace *Get an Adobe ID*.

Se nos abrirá la web de la figura 36, en la que tendremos que introducir nuestro nombre, apellido, email y la contraseña que queremos para la cuenta. La contraseña debe contener al menos un número o símbolo, letras tanto minúsculas como mayúsculas y tener una longitud mínima de 8 carácteres. Además, es imprescindible marcar la opción que dice "*I have read and agree to the Terms of Use and Privacy Policy*", que le indicará al servicio que se han leído y se aceptan los términos de uso y la política de privacidad (ver figura 36).

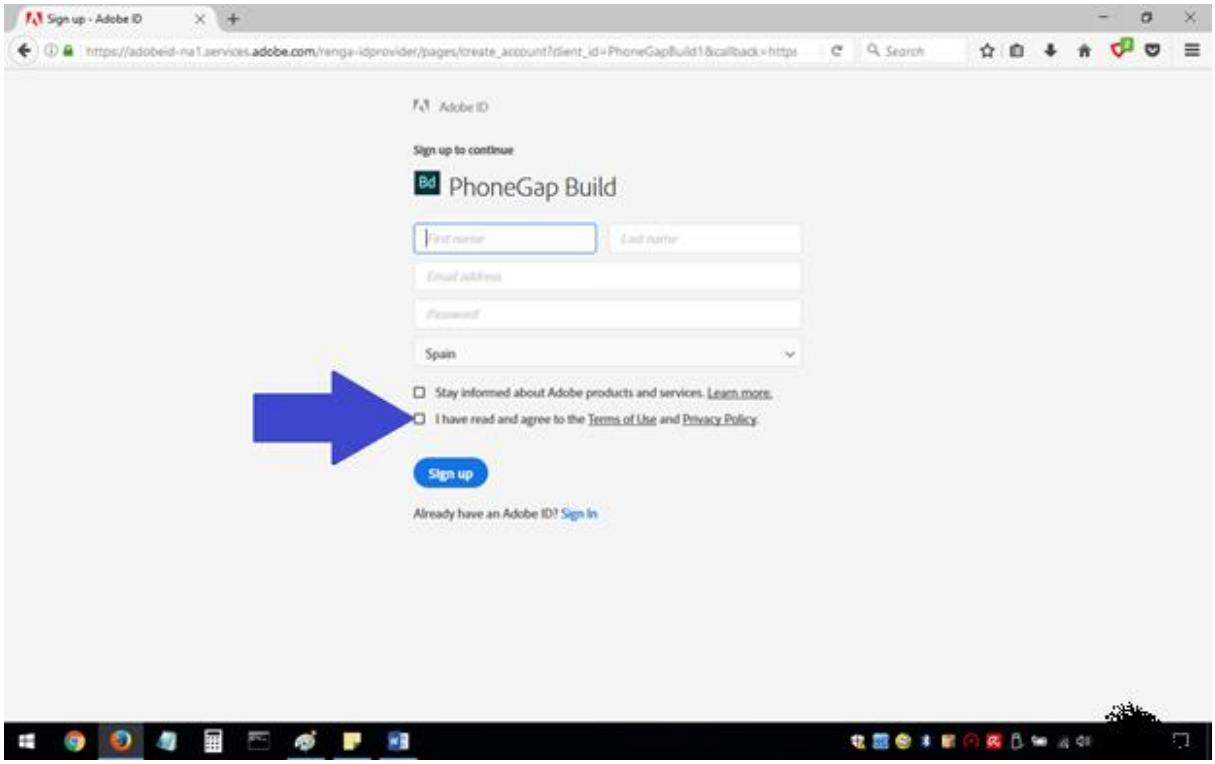

Figura 36: web de registro de un *Adobe ID* o identificador de Adobe para poder usar el servicio *Adobe PhoneGap Build* y ubicación de la opción "*I have read and agree to the Terms of Use and Privacy Policy*".

Una vez cumplimentados los campos nombre, apellido, email y contraseña y cumpliendose los requisitos de esta última, hacemos clic en el botón *Sign up* para crear un nuevo *Adobe ID* o identificador de Adobe necesario para poder usar el servicio *Adobe PhoneGap Build* (ver figura 37).

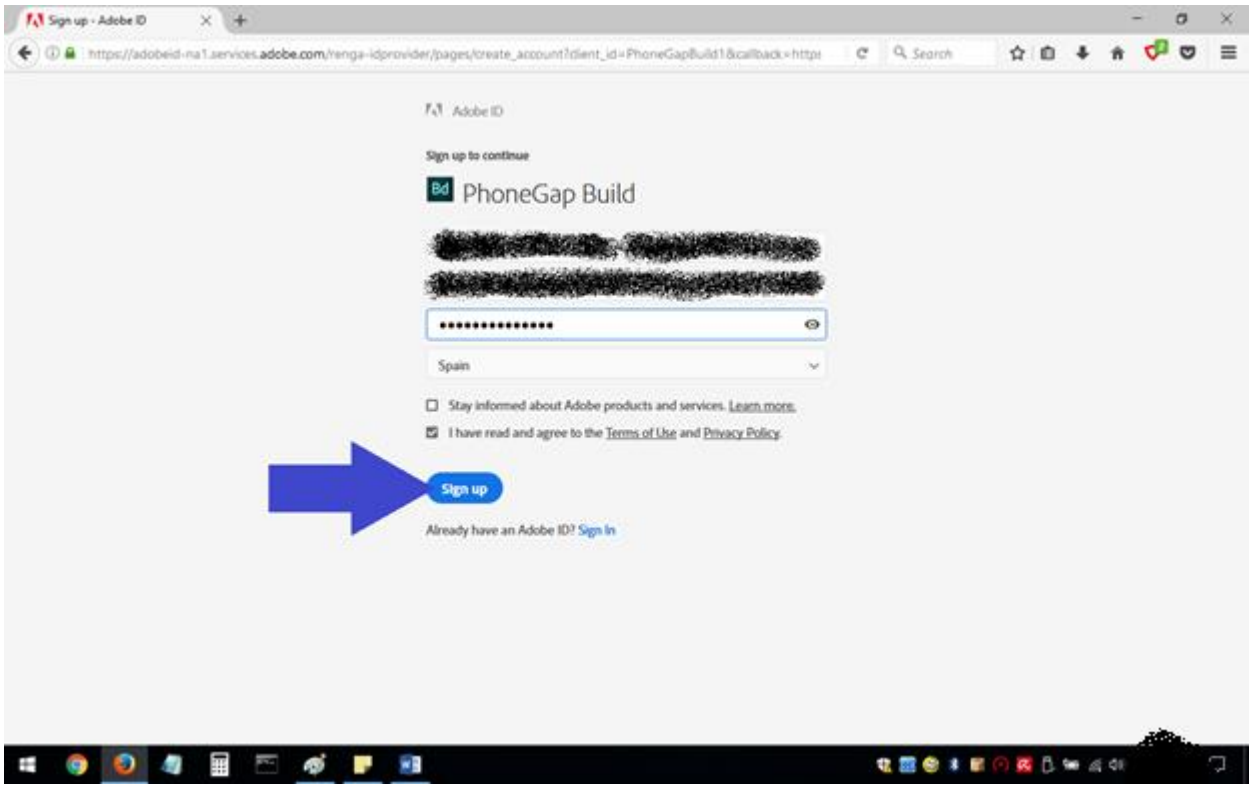

Figura 37: ubicación del botón *Sign up*.

Se nos abrirá la web mostrada en la figura 38, en la que debemos hacer clic en el botón que dice "*Upload a .zip file"*. Al hacer clic en él se nos abrirá una nueva ventana en la que debemos seleccionar el archivo comprimido generado por el entorno de desarrollo *Intel XDK* cuando hicimos clic en el botón *Create Package* de la pestaña *Build* (ver figura 31).

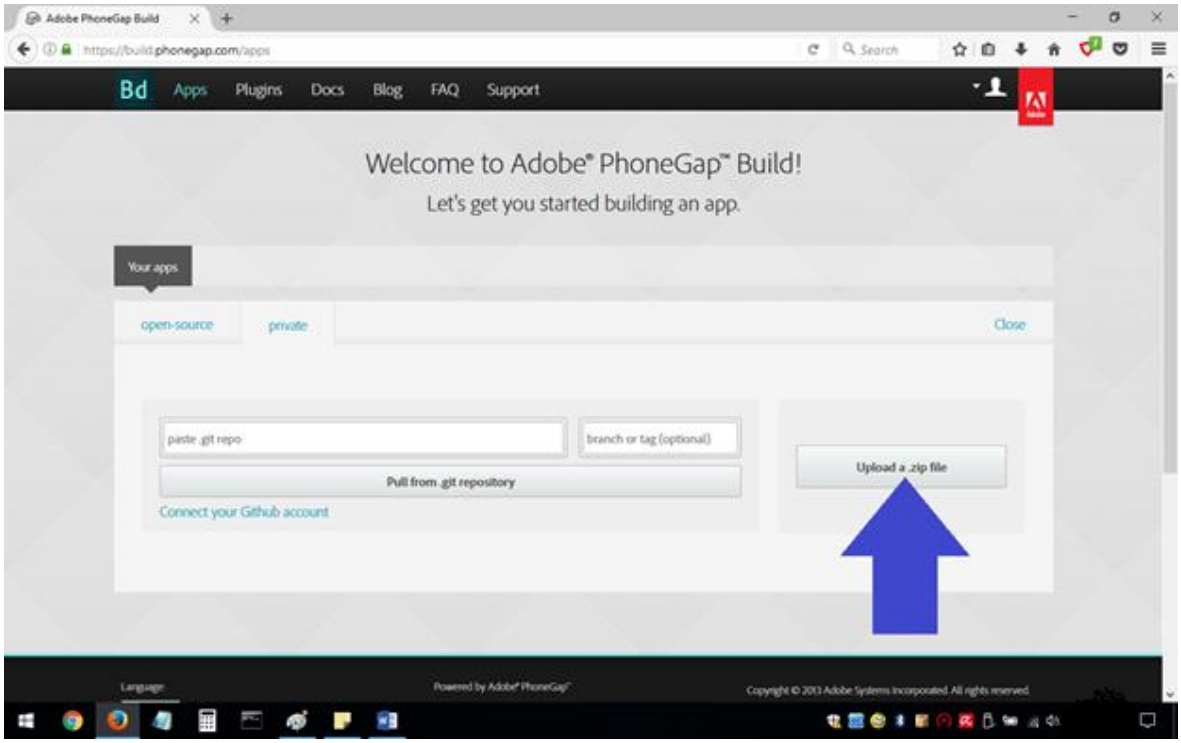

Figura 38: ubicación del botón "*Upload a .zip file*".

Una vez seleccionado el archivo comprimido mencionado anteriormente y que éste haya terminado de transmitirse al servidor, se nos abrirá la web de la figura 39, en la que debemos hacer clic en el botón que dice "*Ready to build*".

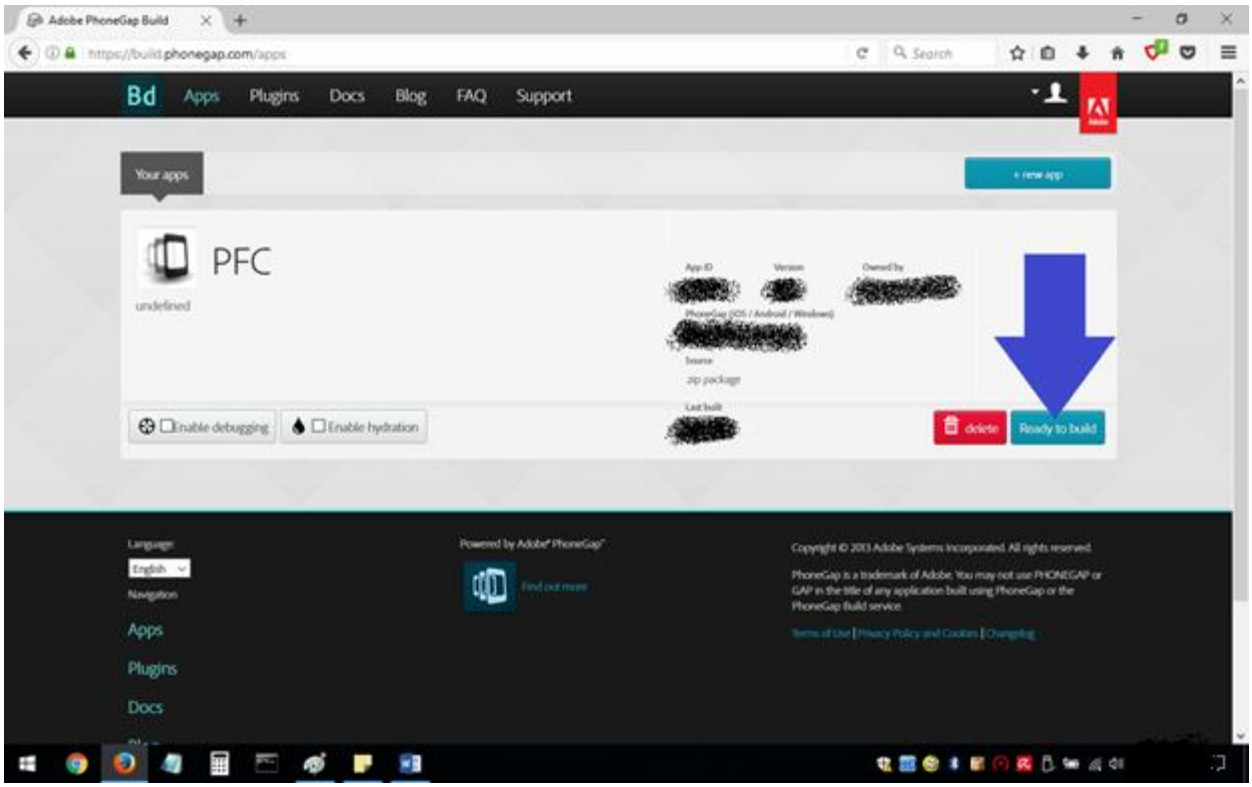

Figura 39: ubicación del botón "*Ready to build*".

El proceso de compilación de la app comenzará, y una vez que haya terminado el aspecto de la web del servicio *PhoneGap Build* será el mostrado en la figura 40. El último paso sería hacer clic en el botón con el símbolo del sistema operativo Android indicado en la figura 40 para descargar el fichero con extensión *apk* que se acaba de generar.

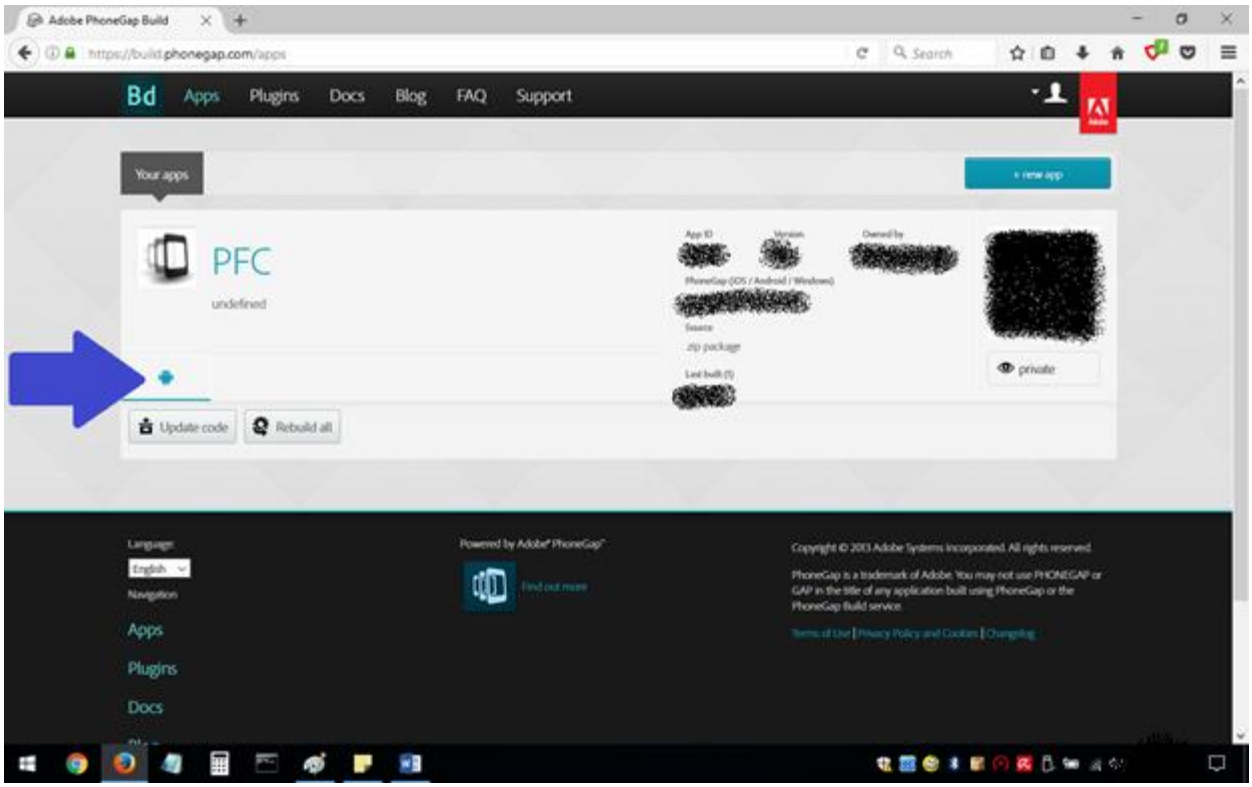

Figura 40: ubicación del botón con el símbolo del sistema operativo Android que provocará la descarga de la app previamente compilada.

ANEXO C: CREACIÓN DE CUENTA EN SERVICIO DE BASE DE DATOS EN LA NUBE

Como se ha ido comentando a lo largo del presente documento, cuando el terminal en el que se ejecuta la app dispone de conexión a Internet la app envía periodicamente los valores de los parámetros obtenidos tantos del vehículo como del propio terminal Android a un servidor de base de datos en la nube cuyos parámetros de configuración deben haberse introducido previamente.

Estos parámetros de configuración del servicio de base de datos en la nube, como es lógico, no aparecen de la nada, sino que están vinculados a la cuenta que el usuario ha creado previamente en el servicio de base de datos en la nube. Se explicarán por tanto a continuación los pasos que hay que llevar a cabo para crear una nueva cuenta de usuario en el servicio de base de datos en la nube que se ha programado en la app, y que por tanto es el que obligatoriamente el usuario debe usar conjuntamente con ella.

El servicio elegido ha sido Zoho Reports, que ofrece unas muy buenas prestaciones en su versión gratuita. Por ejemplo, permite el registro de hasta 1000 filas por día en la tabla de la base de datos que a continuación se explicará cómo crear.

Para crear una nueva cuenta de usuario en Zoho Reports, entraremos al enlace https://www.zoho.com/reports/. En la parte derecha, veremos que aparece un formulario, en el que introduciremos una dirección de correo electrónico y una contraseña, que a continuación confirmaremos (ver figura 41).

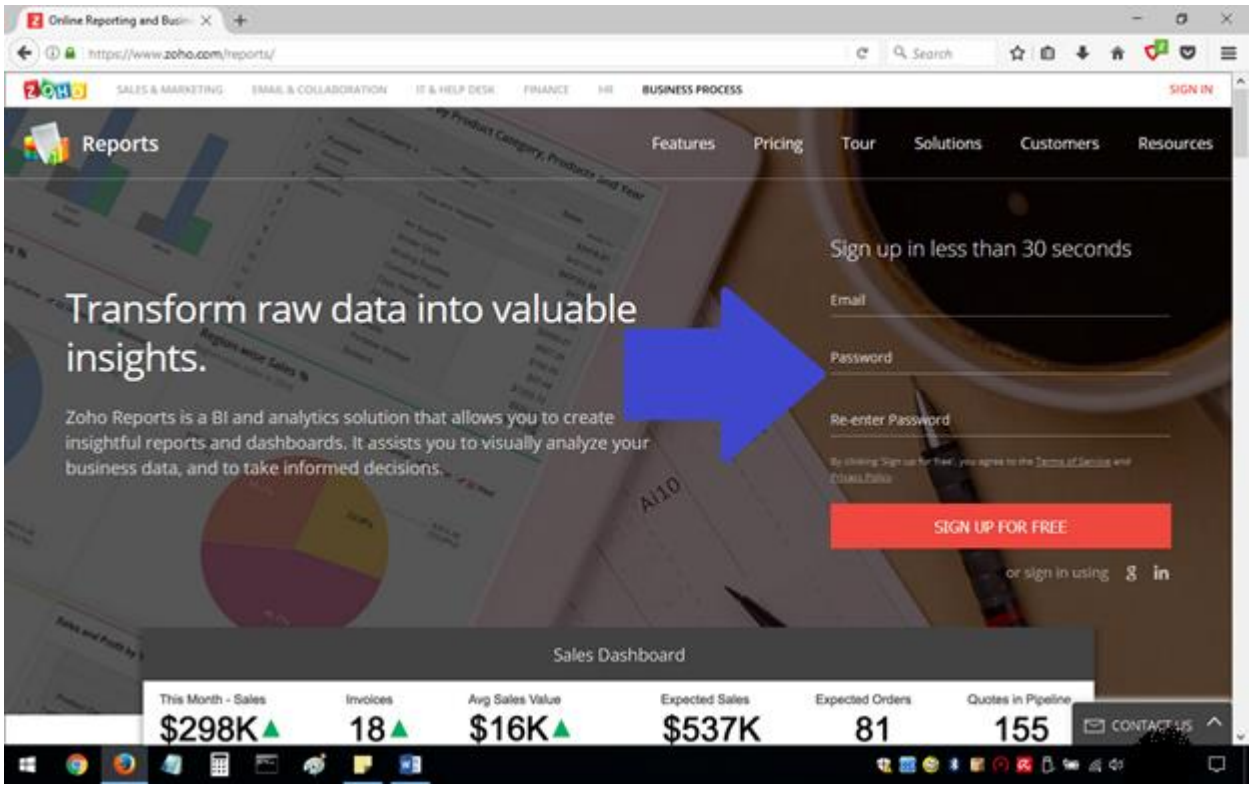

Figura 41: página principal del servicio Zoho Reports, en la que aparece el formulario de registro de una nueva cuenta de usuario.

Una vez introducidos estos datos y cumpliendo los requisitos exigidos a la contraseña haremos clic en el botón que dice "SIGN UP FOR FREE", o lo que es lo mismo, darse de alta gratis, lo que nos llevará a la web mostrada en la figura 42, en la que deberemos hacer clic en el botón "Get Started".

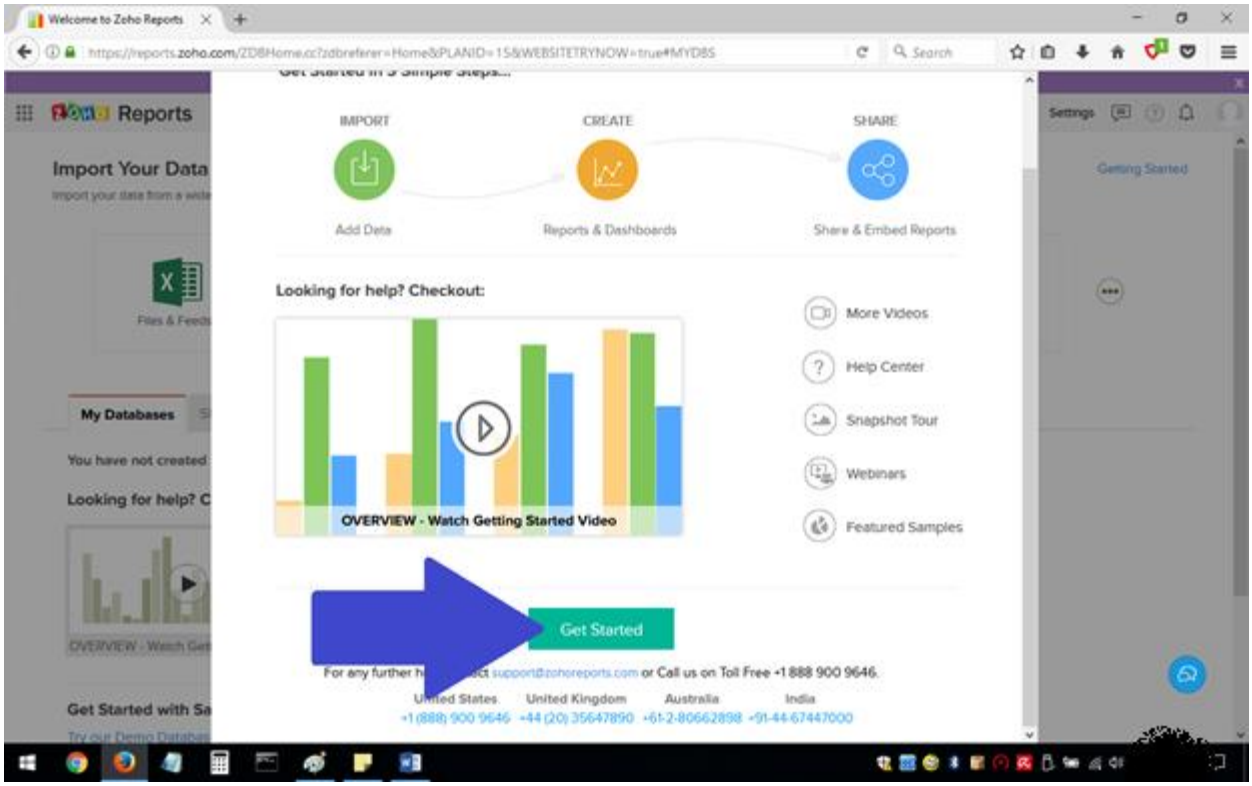

Figura 42: botón "*Get Started*" de la web destino cargada al hacer clic en el botón "*SIGN UP FOR FREE*".

Al hacer clic en el botón "Get Started" se nos mostrará la web de la figura 43, en la que deberemos hacer clic en los tres puntos suspensivos que aparecen a la derecha del todo del apartado "*Import Your Data*":

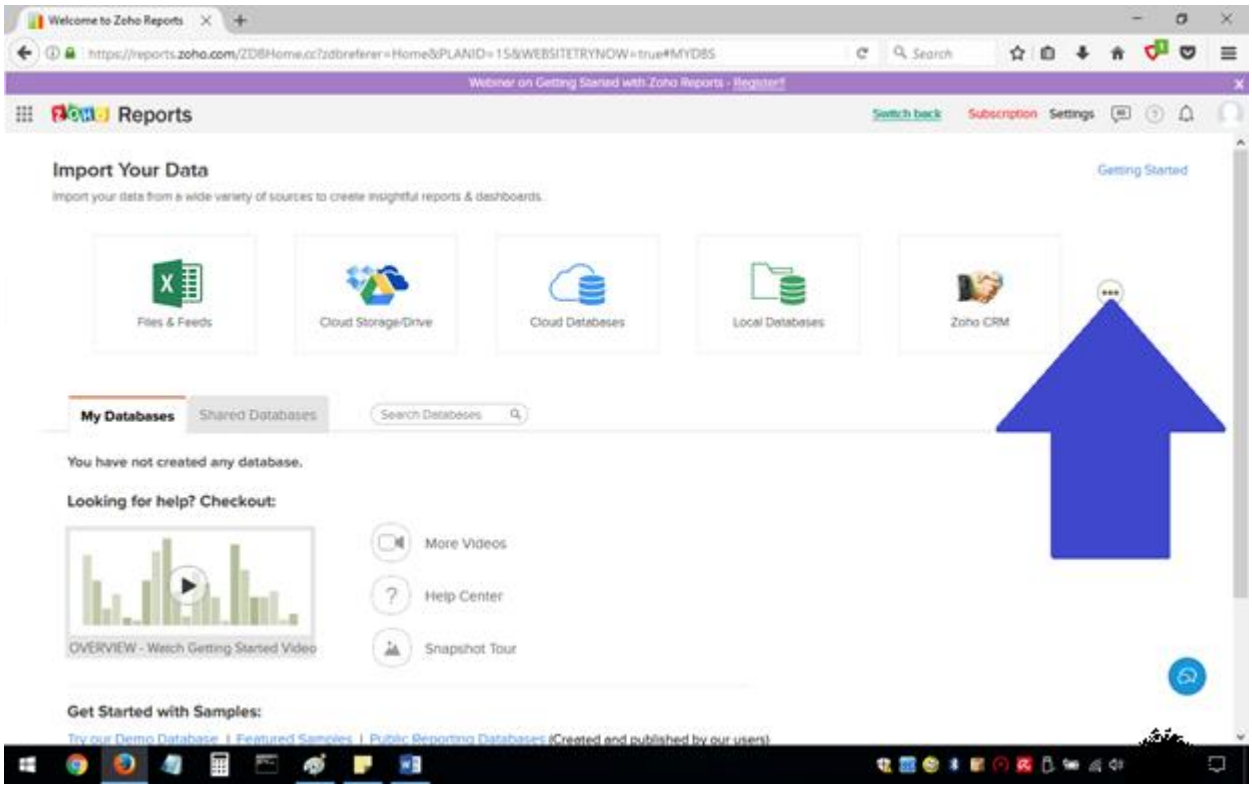

Figura 43: web mostrada al hacer clic en el botón "*Get Started*" de la figura 42.

Al hacer clic en los puntos suspensivos mencionados anteriormente, se mostrarán muchas más opciones de las inicialmente disponibles. Deberemos hacer clic en la que dice "*Blank Database*", o lo que es lo mismo, base de datos en blanco (ver figura 44).

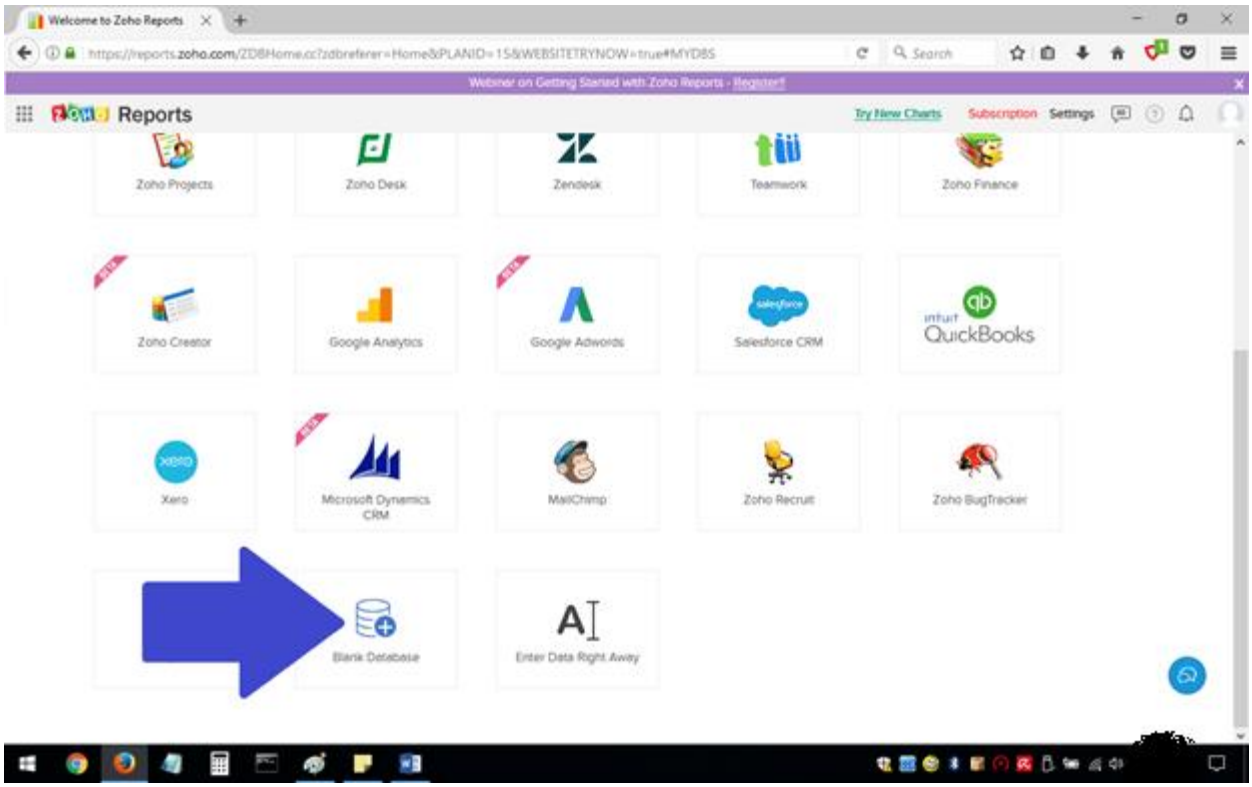

Figura 44: opciones para crear una nueva base de datos.

Se abrirá una especie de ventana dentro de la interfaz de la web en la que tendremos que elegir un nombre para la base de datos, y opcionalmente también una descripción. El nombre que se elija para la base de datos será exactamente el mismo que luego habrá que introducir en la tabla de configuración de la app (ver figura 45).

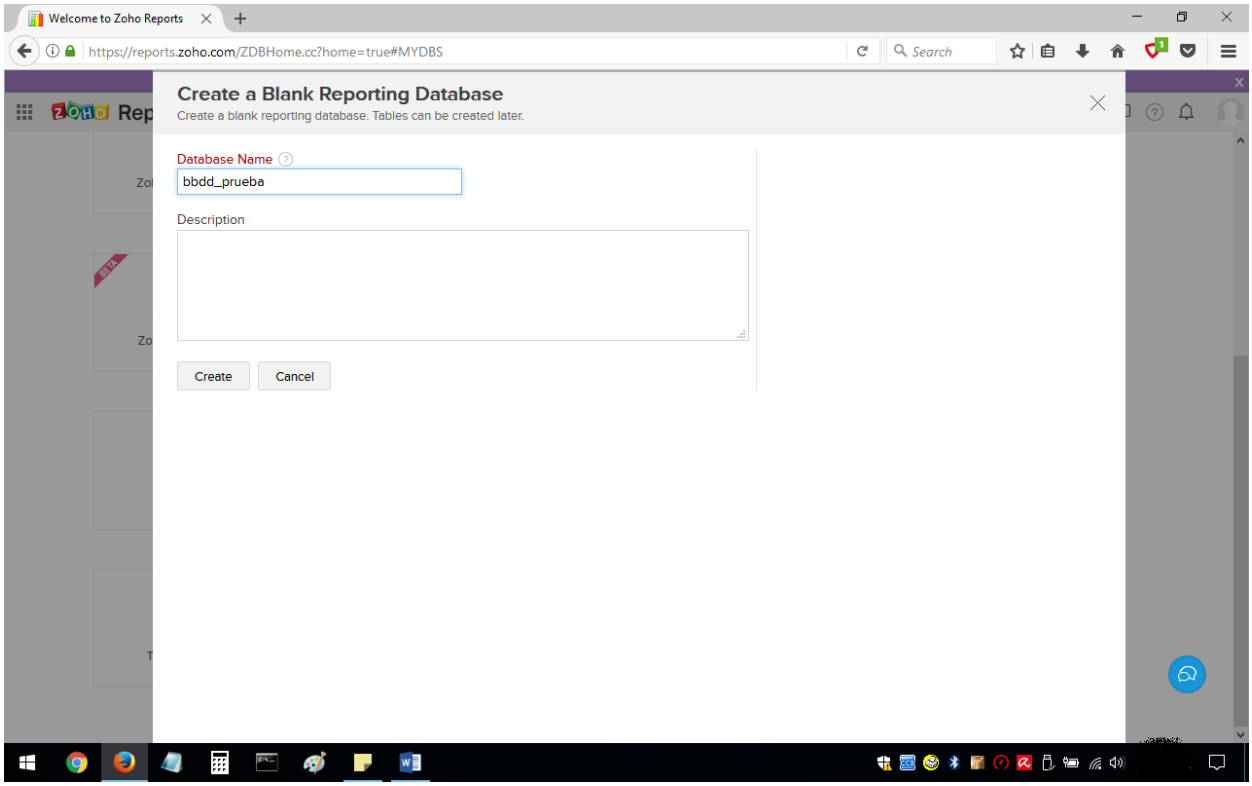

Figura 45: menú de creación de la nueva base de datos. Se opta por el nombre "bbdd_prueba" pero cualquier otro es igual de válido.

Hacemos clic en el botón *Create*, lo que nos llevará a la web mostrada en la figura 46, en la que debemos

hacer clic en el botón que dice "*Import Data / Create New Table*".

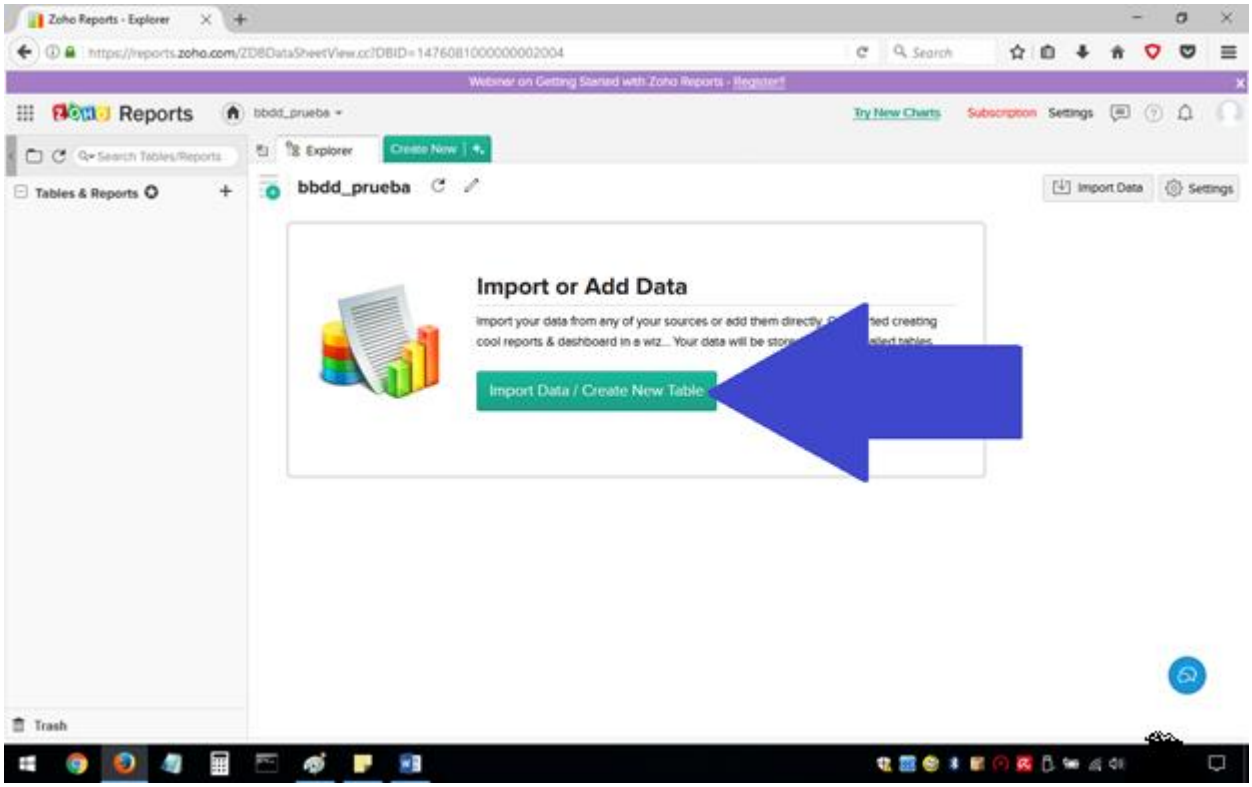

Figura 46: botón "*Import Data / Create New Table*" de la web mostrada al hacer clic en el botón *Create* de la figura 45.

Al hacer clic en el botón "*Import Data / Create New Table*", se nos mostrarán toda una serie de opciones. Debemos elegir una de las últimas, concretamente la que dice "*Enter Data Right Away*" (ver figura 47):

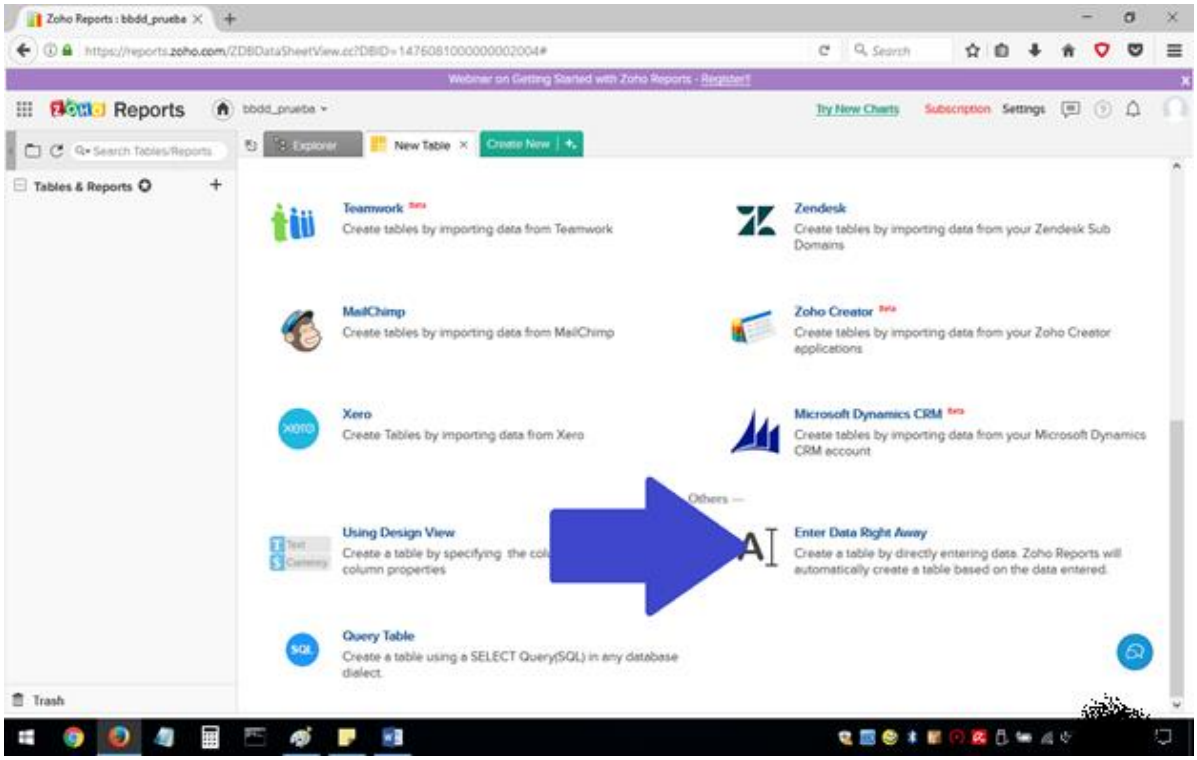

Figura 47: opción "*Enter Data Right Away*" mostrada al hacer clic en el botón "*Import Data / Create New Table*" de la figura 46.

Se nos abrirá la web mostrada en la figura 48. En ella, lo primero que haremos será hacer clic en el botón *Save* para guardar la tabla:

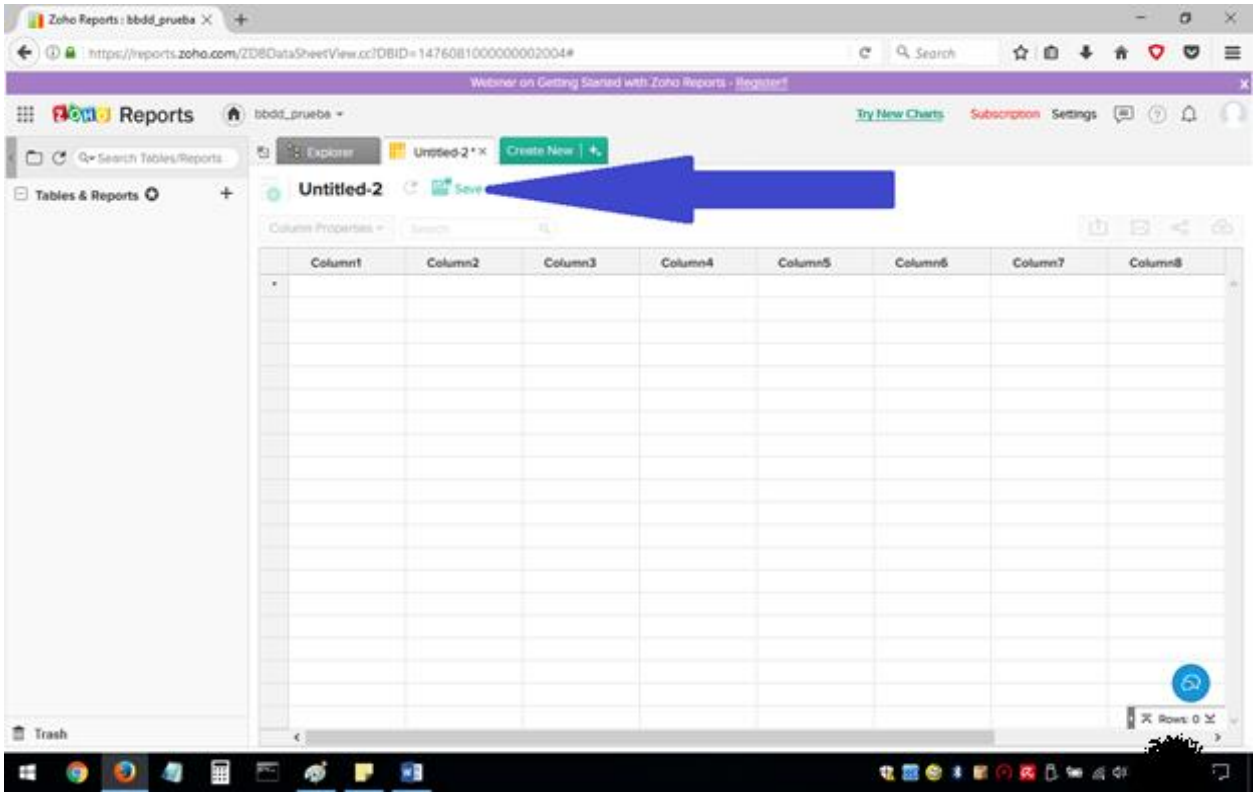

Figura 48: creación de una nueva tabla y ubicación del botón *Save* para guardarla.

Al hacer clic en el botón *Save*, se abrirá un cuadro de dialogo (ver figura 49) en el que tendremos que asignarle un nombre a la nueva tabla (y opcionalmente también una descripción). El nombre que se elija para la tabla será el mismo que posteriormente habrá que introducir en la tabla de configuración de la app.

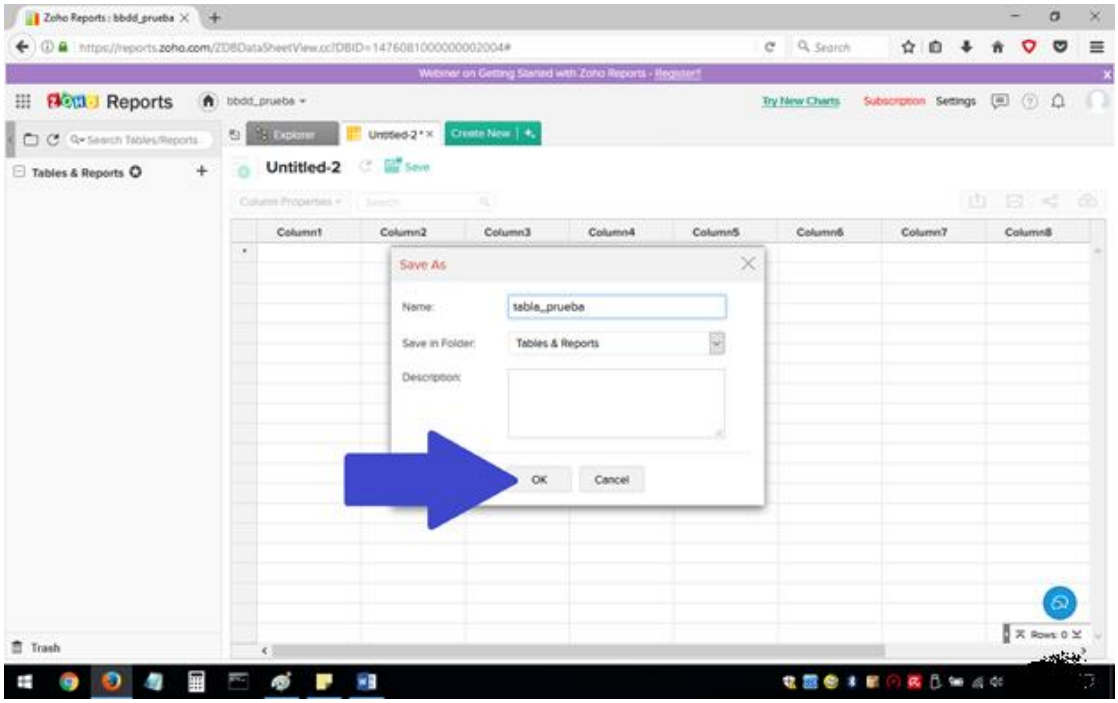

Figura 49: cuadro de dialogo mostrado al hacer clic en el botón *Save* de la figura 48. Se opta por el nombre "tabla_prueba" pero cualquier otro es igual de válido.

Hacemos clic en el botón *OK* para confirmar el nombre de la base de datos, con lo que se mostrará la web de la figura 50, en la que observamos que hay más opciones que las mostradas en la figura 50. Entre las nuevas opciones disponibles está la opción de añadir una nueva columna. Para añadir una nueva columna, pulsaremos el botón *Add* y en el menú desplegable haremos clic en la opción *Add Column*.

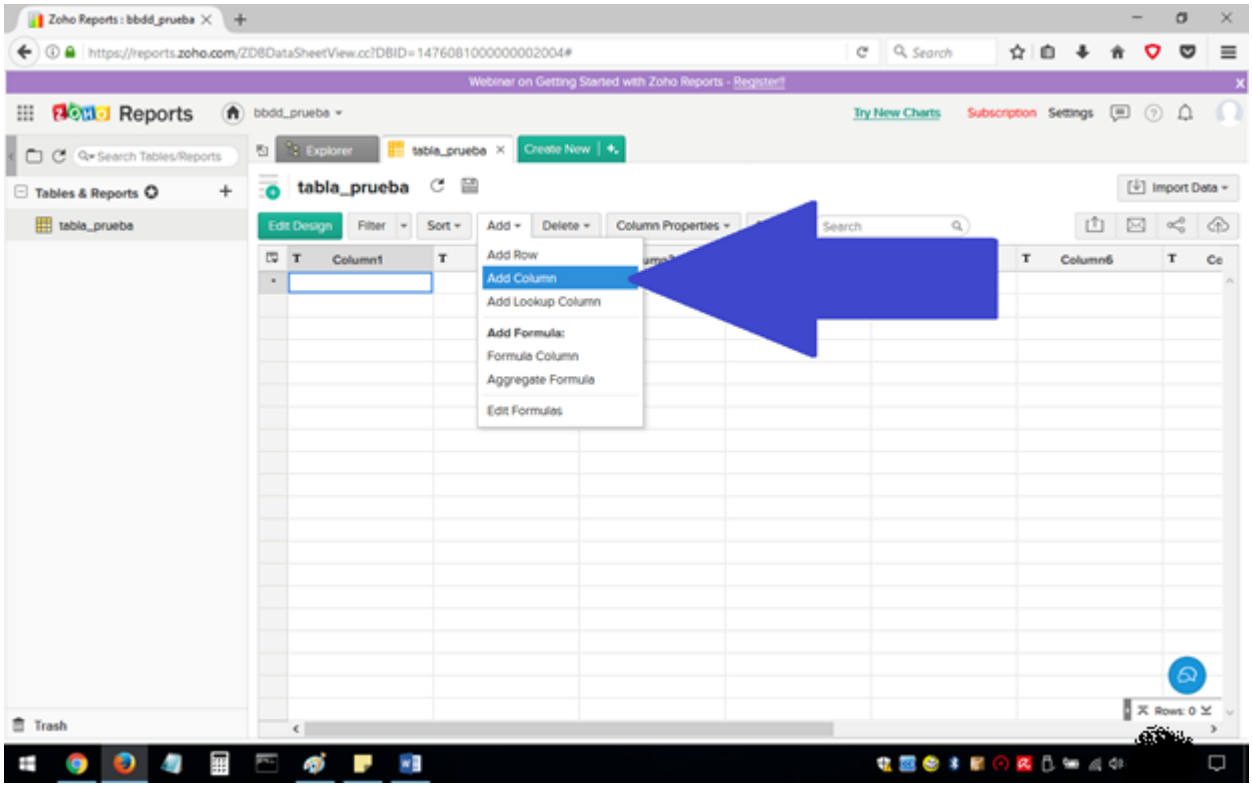

Figura 50: opción de añadir una nueva columna.

Al pulsar en la opción *Add Column* se nos mostrará un cuadro de dialogo en el que simplemente deberemos pulsar *OK* para confirmar el nombre por defecto de la nueva columna (ver figura 51):

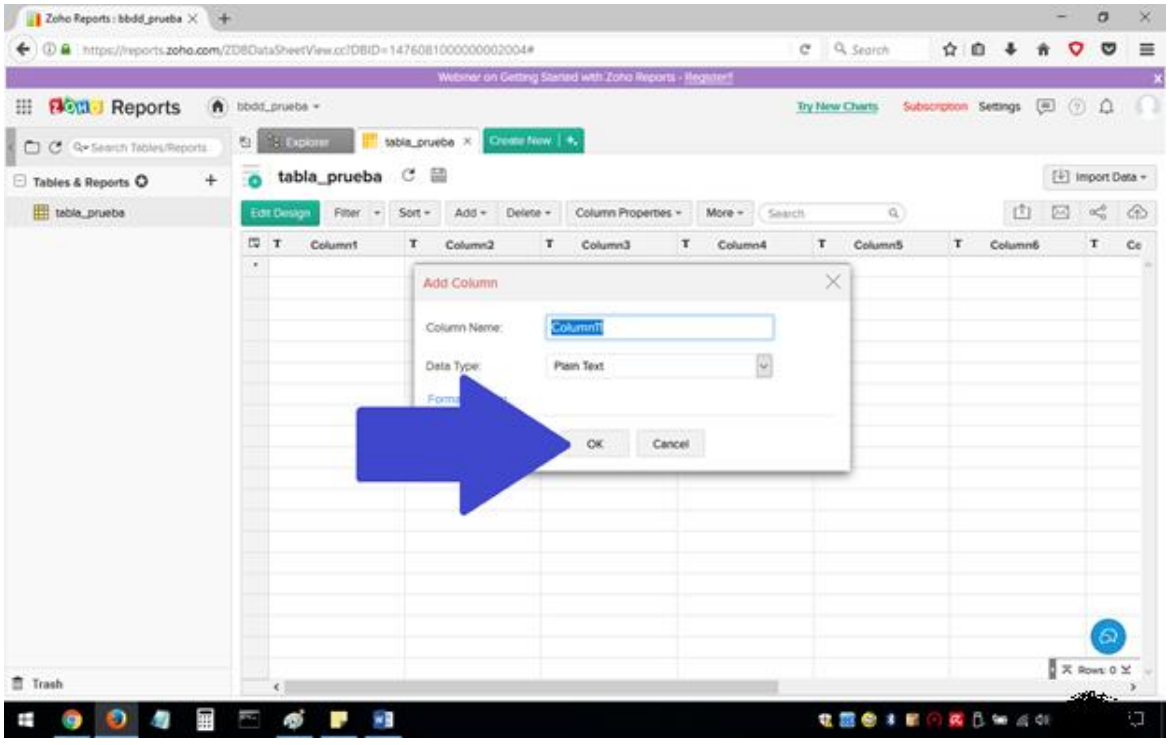

Figura 51: ubicación del botón *OK* del cuadro de dialogo mostrado al hacer clic en la opción *Add Column*.

Al hacer clic en el botón *OK* habremos añadido una nueva columna a la tabla. Habrá que repetir este proceso de añadir una nueva columna tantas veces como sean necesarias hasta que el número total de columnas de la tabla sea 18. Una vez que el número total de columnas de la tabla sea 18, debemos cambiarles el nombre haciendo clic en el título de cada una de ellas, y a continuación en el botón *Column Properties* (ver figura 52).

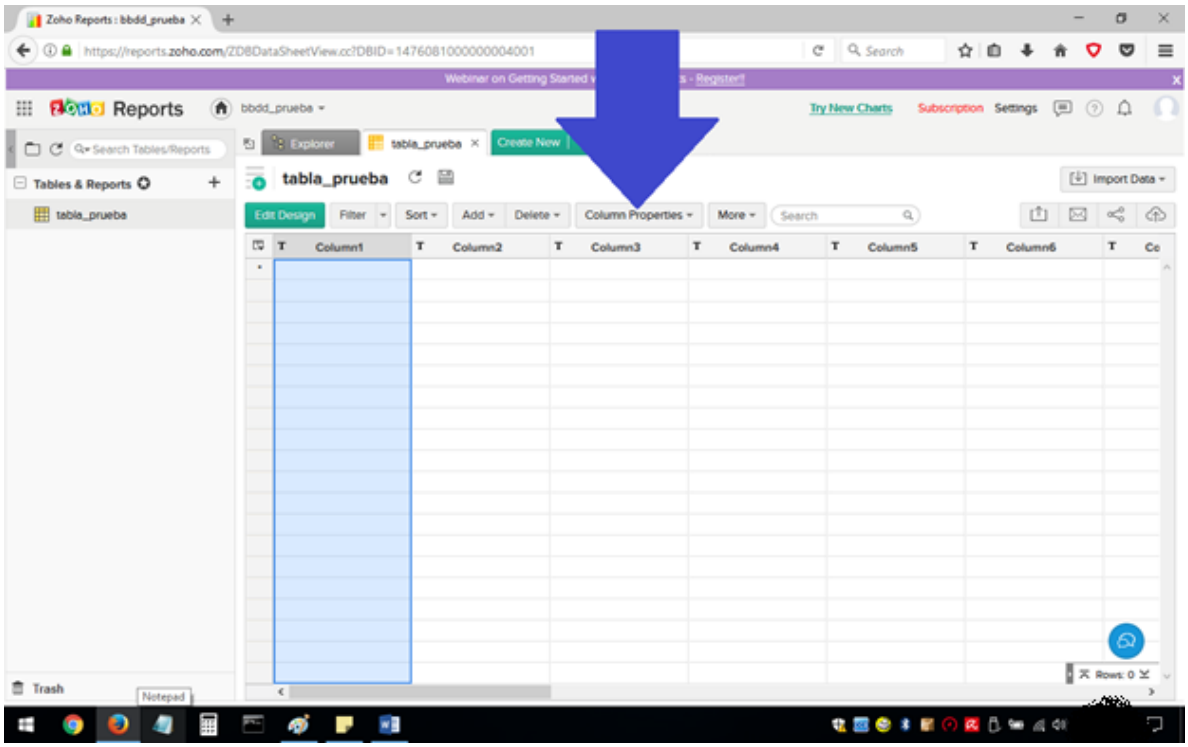

Figura 52: selección de una columna y ubicación del botón *Column Properties*.

Al hacer clic en el botón *Column Properties* se abrirá el menú mostrado en la figura 53, en el que tenemos que seleccionar la opción *Rename Column* para cambiar el nombre de la columna previamente seleccionada.

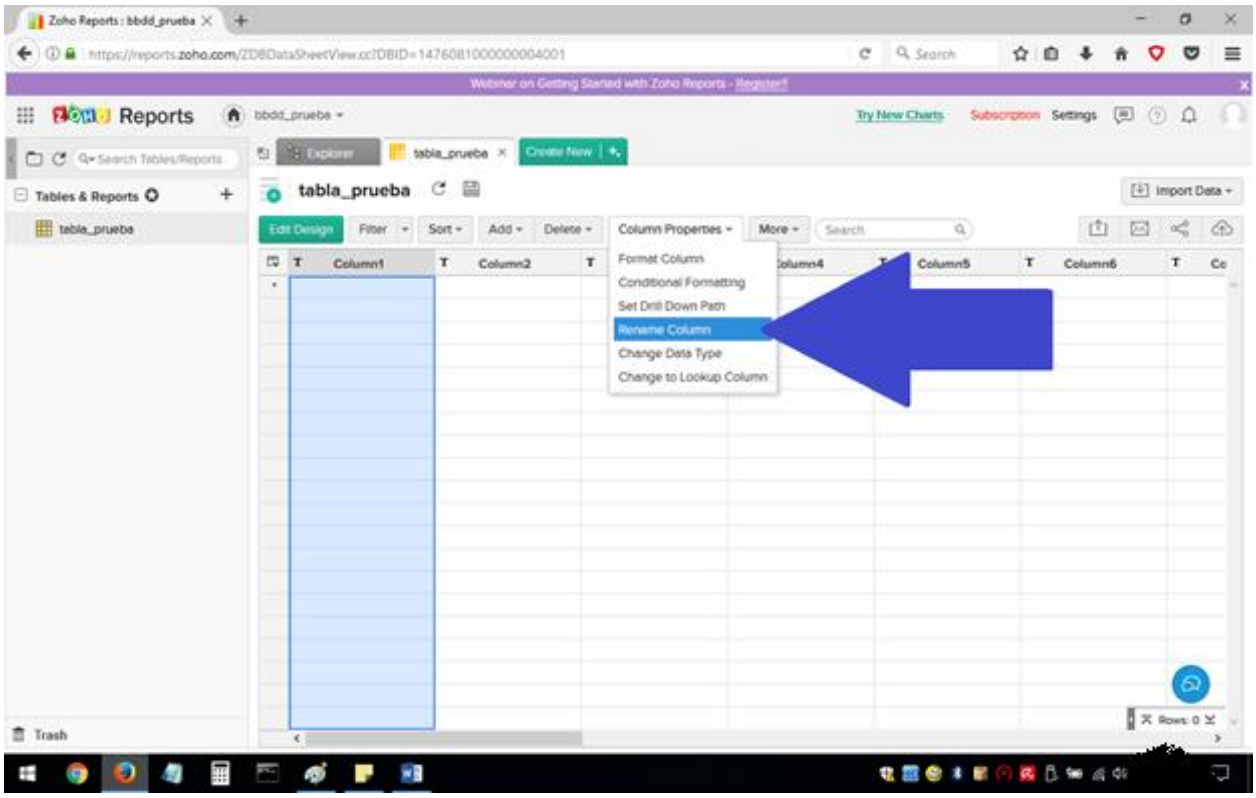

Figura 53: Ubicación de la opción *Rename Column* del menú *Column Properties*.

Al seleccionar la opción *Rename Column* se abrirá el cuadro de dialogo mostrado en la figura 54, en el que podremos escribir el nuevo nombre que deseamos que tenga la columna. Para la primera columna, es decir, la situada más a la izquierda, el nombre debe ser, literalmente, "Número de identificación del vehículo".

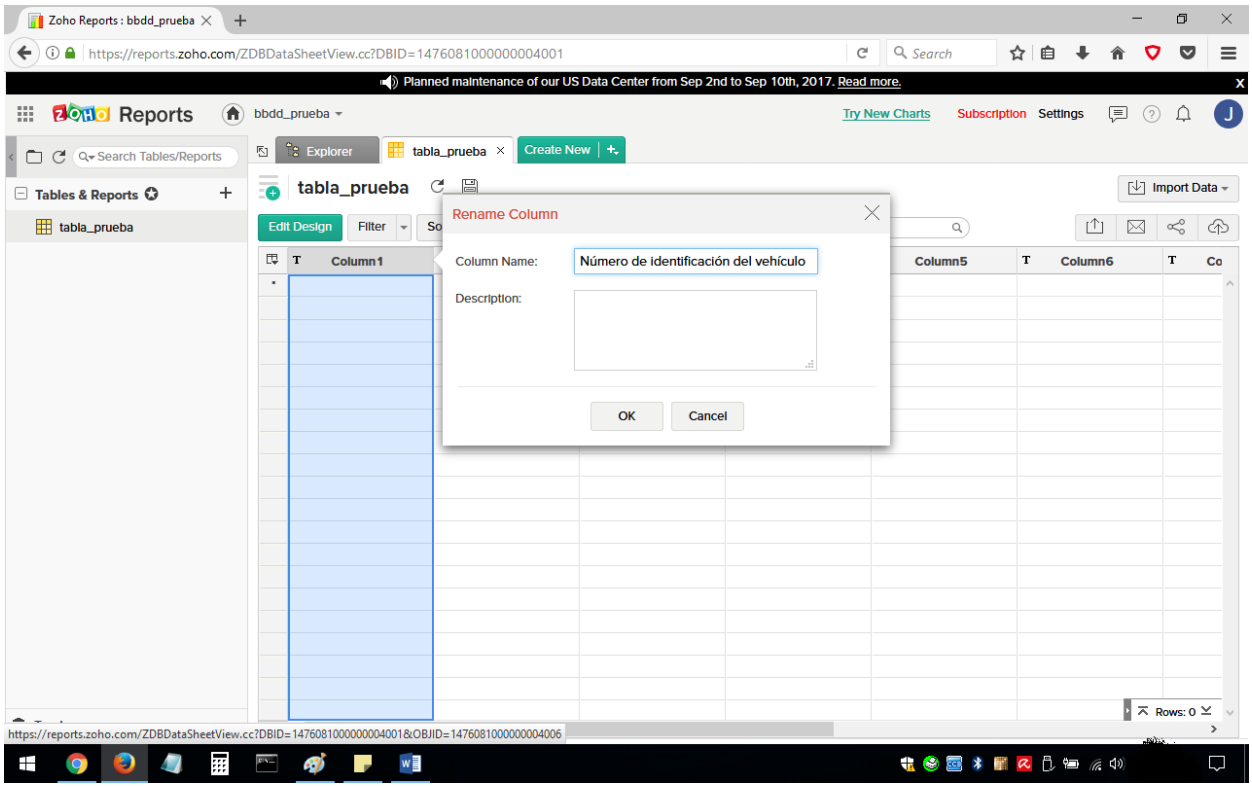

Figura 54: cuadro de dialogo mostrado al seleccionar la opción *Rename Column* del menú *Column Properties*.

Hacemos clic en el botón *OK* del cuadro de dialogo mostrado en la figura 54 y con esto finaliza el cambio de nombre de la primera columna.

Para cambiar el nombre del resto de las columnas habrá que repetir el proceso anteriormente explicado para cada una de ellas. Los nombres para la segunda y siguientes columnas (contando de izquierda a derecha) deberán ser, obligatoriamente, los siguientes:

- 2ª columna: "Fecha".
- 3ª columna: "Hora".
- 4ª columna: "Latitud".
- 5ª columna: "Longitud".
- 6ª columna: "Altitud".
- 7ª columna: "Precisión latitud".
- 8ª columna: "Precisión longitud".
- 9ª columna: "Precisión altitud".
- 10ª columna: "Velocidad".
- 11ª columna: "RPM motor".
- 12ª columna: "Temperatura motor".
- 13ª columna: "Valor velocímetro".
- 14º columna: "Flujo de masa de aire".
- 15ª columna: "Posición acelerador".

- 16ª columna: "Temperatura ambiente".
- 17ª columna: "Número de errores detectados en el vehículo".
- 18ª columna: "Códigos de los errores detectados en el vehículo".

Una vez renombradas las 18 columnas, hacemos de nuevo clic en el botón *Save* mostrado en la figura 48 para guardar los cambios. Ya solo faltaría crear un token de autenticación para que el servidor de la base de datos pueda comprobar que las peticiones HTTP POST que recibirá son de un usuario legítimo.

Para crear un token de autenticación, simplemente debemos acceder a la siguiente URL desde una nueva pestaña o ventana del navegador: https://accounts.zoho.com/apiauthtoken/create?SCOPE=ZohoReports/reportsapi.

Si no se ha cerrado la sesión previamente abierta en Zoho Reports y si no se ha producido ningún error que haya podido impedir que el nuevo token de autenticación se genere correctamente, el resultado mostrado por el navegador serán varias líneas de texto similares a las siguientes que indicarán que se ha generado el nuevo token de autenticación:

```
#


#Wed Jun 29 03:07:33 PST 2013


AUTHTOKEN=bad18eba1ff45jk7858b8ae88a77fa30


RESULT=TRUE
```

El texto mostrado a la derecha del signo igual de la línea que comienza por "AUTHTOKEN" será el que deberemos introducir en el correspondiente campo de texto de la tabla de configuración de la app.

Una vez renombradas las 18 columnas y creado el token de autenticación la configuración y creación de la cuenta del servicio de base de datos en la nube puede darse por concluida. En caso de que el servidor no reciba las peticiones HTTP POST enviadas por la app habría que asegurarse de que los nombres de las columnas son correctos, para lo que habría que prestar especial atención a las letras mayúsculas y las vocales acentuadas. También habría que asegurarse de que tanto el nombre de la base de datos, el nombre de la tabla, el email de la cuenta del usuario y muy especialmente el token de autenticación se han introducido correctamente y sin errores en la tabla de configuración de la app, ya que cualquier discordancia que pudiera haber a este respecto provocaría que el servidor de la base de datos en la nube no recibiera las peticiones HTTP POST enviadas por la app.

ANEXO D: BIBLIOGRAFÍA

[1] http://www.fomento.es/MFOM/LANG_CASTELLANO/DIRECCIONES_GENERALES/TRANSPORTE_TERRESTRE/IGT/desc/TACOGRAFO/registro.htm/

[2] https://es.wikipedia.org/wiki/Tac%C3%B3grafo

[3] http://www.fomento.gob.es/MFOM/LANG_CASTELLANO/DIRECCIONES_GENERALES/TRANSPORTE_TERRESTRE/IGT/TACDIG/TACODIGIT/

[4] http://www.instructables.com/id/Low-Cost-OBD2-Communications-on-K-line-ISO-9141-2-/

[5] http://www.car-angel.es/producto/caja-negra-bbx2-dtw/

[6] https://en.wikipedia.org/wiki/Event_data_recorder

[7] http://www.blackboxpi.com/

[8] https://play.google.com/store/apps/details?id=com.pokevian.skids

[9] https://www.amazon.com/AX5-Vehicle-Tracker-OBDII-Black/dp/B00K1GNWUI

[10] http://www.plusquip.com.au/products/test/eqp-104-obd-11-black-box-recorder/

[11] https://software.intel.com/es-es/intel-xdk

[12] https://cordova.apache.org/

[13] http://stackoverflow.com/questions/25720469/connect-obdsim-to-torqueandroid-app-ubuntu/25763606#25763606

[14] http://www.outilsobdfacile.com/vehicle-list-compatible-obd2/citroen