UNIVERSIDAD CATÓLICA ANDRÉS BELLO.

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIONES

DISEÑO Y DESARROLLO DE UNA APLICACIÓN COMPATIBLE CON DISPOSITIVOS MÓVILES *BLACKBERRY* PARA LA OBTENCIÓN DE INFORMACIÓN ALMACENADA EN SERVIDORES WEB.

REALIZADO POR Henderson León Ramírez Padrón

Carlos Luis Pérez López

TUTOR José Pirrone Puma

FECHA Octubre de 2011

A Nuestros Padres.

Agradecimientos

En primer lugar a Dios y a nuestra alma mater, Universidad Católica Andrés Bello, por haber contribuido a nuestra formación como profesionales en el área de la Ingeniería de Telecomunicaciones.

En el área académica, se pudo contar con la colaboración de reconocidos profesores universitarios, a ellos muchas gracias. Y especialmente a nuestro tutor, quien nos motivó en la temática, y tuvo disposición para brindarnos su apoyo, sugerencias y opiniones ecuánimes; es importante destacar que el valioso aporte de sus conversaciones coadyuvó a llevar de forma amena todo el desarrollo de la investigación.

En lo personal a nuestros padres: Glenys, Luis, Olivia y José por su apoyo incondicional

Al Dr. Inti Garzón y Elides Violeta Padrón MSc, por sus aportes en cuanto a las sugerencias concernientes a la estructuración del Trabajo Especial de Grado desarrollado.

INDICE GENERAL

LISTA DE FIGURAS

RESUMEN

La presente investigación titulada: "Diseño y Desarrollo de una Aplicación Compatible con Dispositivos Móviles BLACKBERRY para la Obtención de Información Almacenada en Servidores Web", se llevó a cabo en la Escuela de Ingeniería de Telecomunicaciones de la Universidad Católica Andrés Bello. El estudio consistió en diseñar una aplicación para ser emulada en simuladores de dispositivos móviles BLACKBERRY, la cual obtendría datos referentes a temperaturas de cavas y posiciones geográficas de vehículos que transportan las mismas, a través de un sistema de información emulado similar al de una empresa dedicada al transporte de alimentos. Para abordar el tema se plantearon varias interrogantes referentes a la estructura que debe poseer la aplicación móvil a diseñar. Posteriormente se procede a recabar información para dar cumplimiento a los objetivos planteados basándose en investigaciones anteriores relacionadas con la temática del Trabajo Especial de Grado, manuales, guías y escritos relacionados con el desarrollo de aplicaciones móviles. Esta investigación dio como resultado la elección de los diversos elementos que conformarían un ambiente de ejecución de pruebas que emula el funcionamiento de la aplicación móvil. Luego de diseñar la aplicación, la misma es sometida a diversas pruebas emulando su funcionamiento al ejecutarla en diferentes simuladores de dispositivos BLACKBERRY, y estableciendo conexiones HTTP con un servidor de aplicaciones previamente configurado para transmitir los datos entre la aplicación móvil y la base de datos responsable de contener la información manejada por la aplicación móvil, apoyándose en una aplicación web ejecutada en dicho servidor. Luego de realizar la emulaciones de la aplicación móvil diseñada en diversos simuladores de dispositivos móviles BLACKBERRY, se logró observar el cumplimiento de los objetivos planteados, tales como: la capacidad de realizar peticiones al servidor y recibir los datos correspondientes a las variables supervisadas, mostrar en imágenes posiciones

geográficas a través de mapas, generar alertas en caso de producirse eventos irregulares relacionados con las temperaturas supervisadas, así como la compatibilidad de la aplicación con los distintos modelos de simuladores escogidos para realizar la emulaciones de la aplicación.

INTRODUCCIÓN

Actualmente la Universidad Católica Andrés Bello lleva en ejecución un proyecto para la Investigación de Nuevas Tecnologías de Información y Comunicaciones (TICs) para procesos de logística de alimentos, dicho proyecto está siendo desarrollado por un grupo de estudiantes y especialistas en Ingeniería Industrial, Telecomunicaciones, Electrónica e Informática. Entre los avances realizados a dicho proyecto, un grupo de estudiantes se encargó de realizar la investigación y las pruebas con Redes Inalámbricas de Sensores que se encargan de realizar mediciones de parámetros (como temperatura, humedad, etc.) que afectan directamente la calidad de los alimentos en el proceso de distribución de los mismos. Por otro lado otro grupo de especialistas se ha encargado de recopilar todas las mediciones realizadas por los sensores y hacer registro de toda esa información en una base de datos a la cual se pueda acceder localmente para supervisar cada uno de los parámetros medidos por los sensores. A partir de este punto surge el problema que dio inicio al presente trabajo de grado y que consiste en cómo acceder a toda esa información resguardada en la base de datos sin necesidad ingresar localmente a ésta.

Este proyecto de trabajo de grado se enmarca en el área de las telecomunicaciones y contempla el diseño y emulación de una aplicación de supervisión basada en lenguaje de programación JAVA para dispositivos móviles, específicamente teléfonos celulares BLACKBERRY con sistema operativo BLACKBERRY O.S. 5.0, la cual permita acceder y supervisar de manera inalámbrica a los diferentes parámetros, registrados en una base de datos, que afectan la calidad de los alimentos durante el proceso de distribución de los mismos.

A partir de la investigación realizada por otro grupo de especialistas que determinaron los parámetros que deben ser tomados en cuenta para supervisar la calidad de los alimentos, en el primer capítulo se expone la importancia de supervisar dichos parámetros tales como la temperatura de las cavas en donde son refrigerados

los alimentos durante el proceso de transporte, así como la posición actualizada de los vehículos en donde son transportados estos alimentos.

En el segundo capítulo se aborda la base conceptual de donde parte el desarrollo de la investigación, se tratan conceptos relacionados con el lenguaje de programación utilizado para el diseño de la aplicación, la Plataforma de Desarrollo de BLACKBERRY y los diferentes elementos necesarios para la ejecución de las pruebas al momento de emular la aplicación diseñada, elementos como el Sistema Administrador de Base de Datos, el Servidor de Aplicaciones, entre otros. Además se exponen los antecedentes encontrados en el uso de dispositivos inalámbricos y Redes de Sensores Inalámbricos para la supervisión de los parámetros que influyen en el proceso de distribución de alimentos.

Luego se presenta la metodología empleada para el desarrollo del presente proyecto, haciendo énfasis en los diferentes elementos a utilizar que conforman el entorno para la ejecución de las pruebas que permitan comprobar el correcto funcionamiento de la aplicación móvil emulada.

En el capítulo de desarrollo se presentan los puntos claves para el diseño de la aplicación, así como de manera esquemática las configuraciones realizadas a las diferentes herramientas que se emplearon para el diseño y emulación de la aplicación y todas sus funciones. Siguiendo con la descripción de los resultados de las diferentes pruebas realizadas para verificar el correcto funcionamiento de la aplicación móvil, y finalmente se exponen las conclusiones y recomendaciones que resultaron de las pruebas realizadas a lo largo del desarrollo del presente proyecto.

CAPITULO I

EL PROBLEMA

1.1 Contextualización y delimitación del Problema.

Actualmente se utilizan sistemas de supervisión inalámbrica que proporcionan información primordial para el control en la producción y almacenamiento de alimentos. Mediante una red de sensores inalámbricos se pueden medir diversas variables, como la temperatura y otros factores que puedan afectar la calidad del producto, en equipos y ambientes de almacenamiento.

De igual forma, es necesario que las labores de supervisión sean en "tiempo real", es decir, que el tiempo que tome el sistema en recolectar información de las características en diferentes instantes, de los elementos controlados por el mismo, sea mínimo. De esta manera se obtendrá la información actualizada de las diferentes variables o parámetros de los cuales depende la calidad del producto.

Un elemento importante a tomar en cuenta para el control de calidad de productos alimenticios, es la temperatura de las cavas en las cuales se almacenan los mismos mientras son entregados a los diferentes centros de distribución (supermercados, abastos, entre otros). Se considera de vital importancia el control de este parámetro, ya que, dependiendo del tipo de alimentos almacenados en estas cavas refrigeradoras, los mismos se conservaran en óptimo estado para un

determinado rango de temperatura. Por esta razón es necesario tener control sobre dicha variable, a fin de informar a la empresa productora sobre la cantidad de productos aptos para el consumo y cuales se deben desechar al no cumplir las normas establecidas para garantizar la conservación óptima de los mismos.

Por otro lado es conveniente tomar en cuenta el posicionamiento actualizado de los vehículos que transportan los productos a los centros de distribución, a fin de tener informado constantemente a la empresa y registrar diferentes datos de tiempos de entrega de los bienes. De esta forma se podrán realizar estudios y proyecciones del proceso de entrega de los alimentos a los distribuidores, y optimizar constantemente dicho proceso, a los fines de hacer el proceso más eficiente y con el menor costo posible.

Basándose en la experiencia de otros servicios de supervisión de parámetros que trabajan sobre *webservers*, se pondrá a prueba el diseño de una aplicación al ser emulada en simuladores de dispositivos móviles (específicamente teléfonos inteligentes *BLACKBERRY).* El usuario encargado de la supervisión de las variables, podrá realizarlo desde cualquier lugar con cobertura de acceso a INTERNET, logrando ingresar a la red de la empresa, y de la misma forma a los servidores que almacenan la información actualizada de las variables que se supervisan con la aplicación.

1.2 Interrogantes de la Investigación.

Los aportes de la aplicación científico-técnica en materia de telecomunicaciones en las últimas décadas del siglo XX y lo que va del siglo XXI, ha sido muy significativo .De allí, que el uso de dispositivos móviles como objeto para llevar adelante sistemas novedosos de comunicaciones requiere de las innovaciones que suelen generarse por lo general en el área académica. Por ello, a los fines de la presente investigación surgen las siguientes interrogantes:

1.2.1 - ¿Qué bases teóricas existen para desarrollar una aplicación que ofrezca datos de temperatura actualizados en cavas de alimentos y que a la vez permita la obtención de datos referentes a las posiciones de estos vehículos cavas?

1.2.2 - ¿Qué rutinas pudieran utilizarse para enviar notificaciones y alertas que pudieran presentarse durante el proceso de transporte de los alimentos en vehículos cavas?

1.2.3 - ¿Cómo desarrollar una interfaz que garantice las labores de supervisión de los usuarios de este servicio?

1.2.4 - ¿Cuáles herramientas utilizar para lograr la factibilidad del diseño del prototipo?

1.2.5 - ¿Cuáles herramientas utilizar para configurar un ambiente de ejecución orientado a emular un sistema que establezca conexiones con la aplicación móvil para suministrarle los datos supervisados por la misma?

1.3 Objetivos de la Investigación.

1.3.1 Objetivo General

Diseñar y emular una aplicación compatible con dispositivos móviles *BLACKBERRY* que permita la supervisión de variables que afectan de manera directa la calidad de productos alimenticios y el proceso de transporte

hacia los establecimientos para su distribución y/o almacenamiento. Así como garantizar con dicha aplicación, un acceso seguro a los servidores que contienen la información perteneciente a dichas variables.

1.3.2 Objetivos Específicos

- Diseñar una aplicación que ofrezca datos de temperatura actualizados de las cavas contenedoras de alimentos. De igual forma debe permitir la obtención de datos referentes a las posiciones de los vehículos encargados de transportar las cavas. Dichos parámetros se obtienen mediante un servidor el cual elabora labores de búsqueda en una base de datos que contienen los valores respectivos de los parámetros mencionados.
- Desarrollar rutinas cuya función este enfocada en enviar notificaciones o alertas en caso de detectarse eventos irregulares (relacionados con las variables supervisadas por la aplicación) durante el proceso de transporte de la mercancía.
- Desarrollar una interfaz para los usuarios responsables de la supervisión de las variables que ofrezca de manera práctica la información de los parámetros en cuestión.
- Implementar herramientas de seguridad relacionada directamente con la aplicación, que garantice exclusividad de acceso solo a usuarios autorizados para obtener la información de las variables, así como la confidencialidad de la información obtenida por la aplicación móvil haciendo uso de estándares criptográficos.
- Configurar un ambiente de ejecución que permita emular la manera en que se suministra a la aplicación móvil datos relacionados con las variables supervisadas por la misma, con el fin de comprobar su correcto funcionamiento.

1.4 Justificación.

La presente investigación es de carácter descriptiva y su diseño es experimental, se lleva a cabo como requisito para optar al grado de Ingeniero en Telecomunicaciones en la Universidad Católica Andrés Bello (UCAB), los hallazgos en esta investigación pueden servir de referencia para investigaciones similares y relacionadas con esta área del conocimiento, tanto a nivel académico como industrial, toda vez que el desarrollo de un prototipo es de relevancia tecnológica y económica.

Adicionalmente, el alcance de la misma es concebido como un aporte para futuras investigaciones en esta área del conocimiento ,como texto de consulta en la Escuela de Ingeniería de Telecomunicaciones de la UCAB, que podría posibilitar la gestión de conocimiento y experticia en el sector de las telecomunicaciones. Así mismo dentro de las limitaciones, los autores consideran pertinente señalar la ausencia del servicio *BLACKBERRY Maps* en Venezuela, por lo que no es factible usar las librerías referentes a posicionamiento contenidas en el JRE (*Java Runtime Enviroment*) utilizado para el desarrollo de la aplicación. Por otro lado, existe una limitante en cuanto al sistema operativo propio del dispositivo en donde se ejecuta la aplicación, ya que la aplicación es compatible solo con *BLACKBERRY* OS5. Otra limitación encontrada, es la posibilidad de implementar la aplicación diseñada en dispositivos *BLACKBERRY*, ya que la misma pretende utilizar *API's* caracterizadas por permitir el uso de criptografía y almacenamiento de datos en la memoria del dispositivo. Las mismas requieren una firma digital paga (*Singing Key*) para la ejecución en un dispositivo real, y la misma es proporcionada por la empresa fabricante de estos dispositivos (*Research In Motion*) (Rizk, 2009, pág. 9).

CAPITULO II

MARCO TEÓRICO.

En este capítulo se trae a colación lo referente a las bases conceptuales relacionadas con el desarrollo de aplicaciones en *BLACKBERRY*, tecnologías y lenguaje que permitieron el desarrollo de la presente investigación, considerándose de especial énfasis: las Redes WSN, al Lenguaje de Programación JAVA, los Servidores de Aplicaciones, la Plataforma de Desarrollo *BLACKBERRY* y el Sistema Administrador de Base de Datos.

2.1 Redes WSN.

El desarrollo de una aplicación para teléfonos BLACKBERRY que permita acceder a información contenida en servidores, requiere dar a conocer la tecnología de red que se implementa para recolectar y llevar toda esa información a los servidores, dicha tecnología se conoce como Redes Inalámbricas de Sensores o WSN (*Wireless Sensor Network*).

Según (García & De Sousa, 2011), las redes WSN se basan en dispositivos de bajo costo y consumo, llamados nodos, los cuales son capaces de obtener información de su entrono, procesarla y luego comunicarla a través de enlaces inalámbricos hasta un nodo central. Estos nodos forman parte de la infraestructura de comunicación reenviando los mensajes transmitidos por nodos más lejanos hacia el centro de coordinación.

Estas redes de sensores están formadas por dispositivos inalámbricos distribuidos espacialmente, los cuales utilizan sensores para controlar diversas condiciones como temperatura, presión, humedad, movimiento, entre otros. En lo referido a este proyecto, los sensores se encontrarían distribuidos en las cavas de refrigeración de los camiones monitoreando variable como la temperatura y la ubicación de dichos camiones. En la Figura 1 se observa como seria la distribución de los dispositivos inalámbricos y la topología de red que se implementaría.

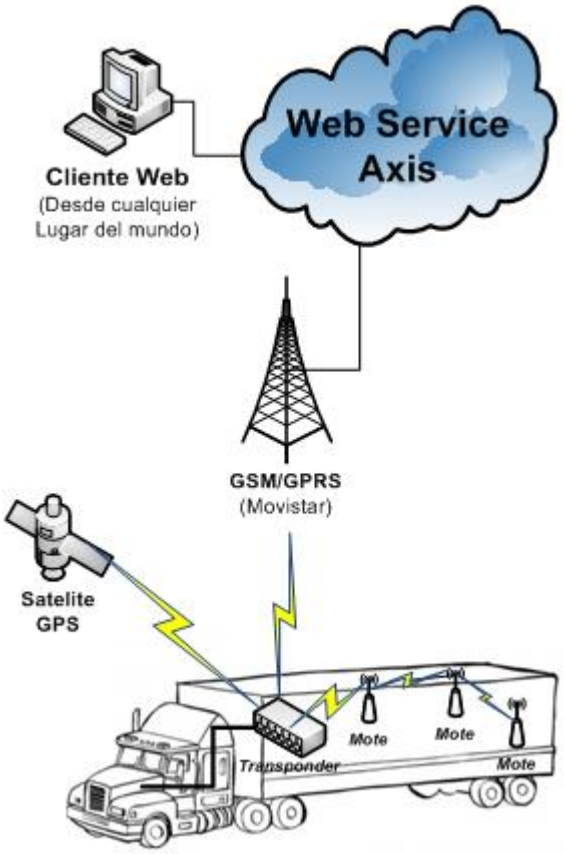

Figura 1. Sistema Integral para la Gestión y Monitoreo de Camiones cavas de alimentos.

Fuente: (García & De Sousa, 2011)

2.2 Elementos de una Red de Sensores Inalámbricos (WSN)

Según **(Wiley, 2004)**, Los elementos están integrados por: sistema de adquisición de datos, nodo, Redes de Sensores (*Mota*), *Gateway y* Estación Base.

2.2.1 Sistemas de Adquisición de Datos

Son los sistemas conformados por sensores que toman del medio la información (temperatura, humedad, etc.) y la transforman en señales eléctricas. En el mercado existe una gran variedad de placas de sensores que pueden medir diversos parámetros, como presión, GPS, intensidad de luz, humedad, temperatura, etc.

2.2.2 Nodo

Estos dispositivos son unidades autónomas compuestas por un microcontrolador, una fuente de energía, un radio-transceptor (RF) y un elemento sensor. Básicamente sus componentes son:

- Una CPU
- Memoria Flash
- Memoria separada para datos programas
- La Placa de Sensores: luz, presión, etc.
- Radio para comunicarse con otras motas
- Conversor analógico-digital (ADC)
- Baterías

2.2.3 Redes de Sensores (Mota)

Cada nodo de la red consta de un dispositivo con micro-controlador, sensores y transmisor/receptor, y forma una red con muchos otros nodos, también llamados *motas* o sensores. Por otra parte, un sensor es capaz de procesar una limitada cantidad de datos. Pero cuando se coordina la información entre un importante número de nodos, éstos tienen la habilidad de medir un medio físico dado con gran detalle. Con

todo esto, una red de sensores puede ser descrita como un grupo de *motas* que se coordinan para llevar a cabo una aplicación especifica. Al contrario que las redes tradicionales, las redes de sensores llevarán con más precisión sus tareas dependiendo de lo denso que sea el despliegue y lo coordinadas que estén.

En los últimos años, las redes de sensores han estado formadas por un pequeño número de nodos que estaban conectados por cable a una estación central de procesado de datos. Hoy en día, sin embargo, estos diseños se enfocan mayormente en redes de sensores distribuidas e inalámbricas. La razón por la cual son distribuidas e inalámbricas, reside en que cuando la localización de un fenómeno físico es desconocida, este modelo permite que los sensores estén mucho más cerca del evento de lo que estaría un único sensor.

Además, en muchos casos, se requieren muchos sensores para evitar obstáculos físicos que obstruyan o corten la línea de comunicación. El medio que va a ser monitorizado no tiene una infraestructura, ni para el suministro energético, ni para la comunicación. Por ello, es necesario que los nodos funcionen con pequeñas fuentes de energía y que se comuniquen por medio de canales inalámbricos.

Otro requisito para las redes de sensores será la capacidad de procesamiento distribuido. Esto es necesario porque, siendo la comunicación el principal consumidor de energía, un sistema distribuido significará que algunos sensores necesitarán comunicarse a través de largas distancias, lo que se traducirá en mayor consumo. Por ello, es una buena idea el procesar localmente la mayor cantidad de energía, para minimizar el número de bits transmitidos.

En la Figura 2 se puede observar un dispositivo *Mota* con su Placa de Sensores y componentes.

Figura 2. Dispositivo Mota.

Fuente: (Wiley, 2004)

2.2.4 *Gateway*

El *Gateway* es el dispositivo que dentro de una red WSN ejerce la función como nodo central y permite la interconexión entre la red de sensores y una red TCP/IP.

2.2.5 Estación Base

Consiste en un recolector de datos basado en un ordenador común o servidor. En la Figura 3 puede observarse una distribución de una red WSN en donde los dispositivos *Mota* que toman mediciones de temperatura, humedad, etc., envían los datos a un *Gateway* que se encarga luego de transmitir la información a la Estación Base en donde se van recolectando los datos.

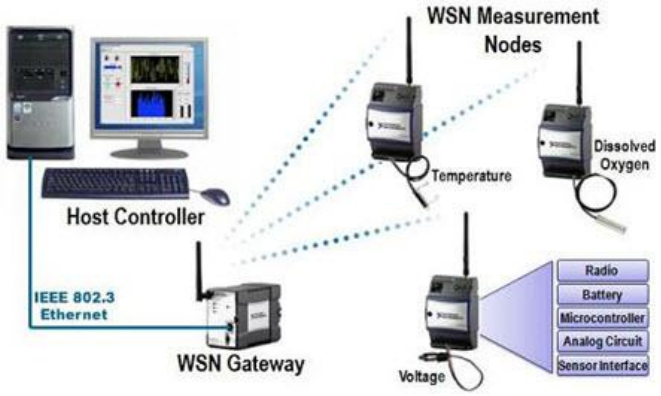

Figura 3. Arquitectura de Red WSN. Mota, Gateway y Estación Base.

Fuente: (Wiley, 2004)

2.3 Lenguaje de Programación *JAVA*.

Es pertinente dar una breve descripción del lenguaje de programación responsable de la ejecución de instrucciones que llevan a cabo la el funcionamiento del sistema.

JAVA es un lenguaje de programación orientado a objetos, el mismo fue desarrollado originalmente por *SUN MICROSYSTEMS* a principios de los años 90. El lenguaje presente una sintaxis similar a *C y C++*, y a diferencia de los lenguajes mencionados anteriormente el mismo ofrece un modelo de creación de objetos más simple ya que el lenguaje no emplea herramientas de bajo nivel que sistemáticamente proporcionan errores al compilar y ejecutar programas, como la manipulación directa de punteros o memoria del computador responsable de la ejecución del programa[1]. El lenguaje presenta como características principales:

[1] http://es.wikipedia.org/wiki/JAVA_(lenguaje_de_programaci%C3%B3n).

2.3.1 Orientado a objetos (“OO”)

Hace referencia al método de programación usado por el lenguaje y al diseño del lenguaje mismo. Viéndose el mismo desde una perspectiva general, se puede afirmar que el mismo consiste en diseñar programas de forma que los distintos tipos de datos que usen estén unidos a sus operaciones. De esta forma los códigos de ejecución y los datos referentes al programa se enfocan como un todo en lo que se conoce como un objeto. Un objeto puede verse como un paquete que contiene el “comportamiento” (el código) y el “estado” (datos). El principio es separar aquello que cambia de las cosas que permanecen inalterables. Usualmente al cambiar la estructura de datos, implica un cambio inmediato en el código que realiza operaciones con dichos datos o viceversa.

La separación de estos dos elementos que conforman el objeto genera un ambiente con mayor estabilidad en la plataforma en cual se desarrollan los programas. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del *software* entre proyectos, una de las premisas fundamentales de la Ingeniería del *Software*. Un objeto genérico “cliente”, por ejemplo, debería en teoría tener el mismo conjunto de comportamiento en diferentes proyectos, sobre todo cuando estos coinciden en cierta medida, algo que suele suceder en las grandes organizaciones. En este sentido, los objetos podrían verse como piezas reutilizables que pueden emplearse en múltiples proyectos distintos, posibilitando así a la industria del *software* a construir proyectos de envergadura empleando componentes ya existentes y de comprobada calidad; conduciendo esto finalmente a una reducción drástica del tiempo de desarrollo.

2.3.2 Independencia de la Plataforma

Gosling, Steele, & Bracha (2005), indican que esta propiedad hace referencia a una característica de gran importancia, ya que la misma sugiere la correcta operación de los programas escritos en este lenguaje en diferentes *hardwares* (siempre y cuando los mismos posean el soporte para dicho lenguaje). Este es el significado de ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de JAVA, *"Write Once, Run Anywhere".*

Para ello, se compila el código fuente escrito en lenguaje JAVA, para generar un código conocido como "*bytecode*" (específicamente JAVA *bytecode* las cuales son instrucciones máquina simplificadas específicas de la plataforma JAVA). Esta pieza está "a medio camino" entre el código fuente y el código máquina que entiende el dispositivo destino. El *bytecode* es ejecutado entonces en la máquina virtual (JVM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su *hardware*), que interpreta y ejecuta el código. Además, se suministran bibliotecas adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o *threads*, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el *bytecode* generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (*Just In Time*) (Gosling, Steele, & Bracha, 2005).

Las primeras implementaciones del lenguaje usaban una máquina virtual interpretada para conseguir la portabilidad. Sin embargo, el resultado eran programas que se ejecutaban comparativamente más lentos que aquellos escritos en C o C++. Esto hizo que JAVA se ganase una reputación de lento en rendimiento. Las implementaciones recientes de la JVM (*JAVA Virtual Machine*) dan lugar a programas que se ejecutan considerablemente más rápido que las versiones antiguas, empleando diversas técnicas, aunque sigue siendo mucho más lento que otros lenguajes.

El concepto de independencia de la plataforma de JAVA cuenta, sin embargo, con un gran éxito en las aplicaciones en el entorno del servidor, como los Servicios Web, los *Servlets*, los *JAVABeans*, así como en sistemas empotrados basados en *OSGi*, usando entornos JAVA empotrados.

2.3.3 Recolector de Basura (*Garbage Collector*).

Representa la solución que aporta el lenguaje al control de fugas de de memoria durante la ejecución. Evidentemente el programador es responsable de definir la creación de los objetos, sin embargo, el entorno de ejecución del lenguaje (*JAVA Runtime*) es el encargado de gestionar el ciclo vida de los objetos definidos. Profundizando en mayor el grado la funcionabilidad del recolector, se puede asumir que el programa, u otros objetos pueden tener localizado un objeto mediante una referencia a éste. Si se detecta un objeto que se ha quedado sin referencias, el recolector de basura elimina el objeto, de esta forma se libera el espacio que anteriormente era ocupado por el objeto en la memoria del (ejemplo: un objeto creado y únicamente usado dentro de un método sólo tiene entidad dentro de éste; al salir del método el objeto es eliminado). Aun así, es posible que se produzcan fugas de memoria si el código almacena referencias a objetos que ya no son necesarios(es decir, pueden aún ocurrir, pero en un nivel conceptual superior. Como conclusión, el recolector de basura de JAVA ofrece una fácil creación y eliminación de objetos.

Sun Microsystem ha desarrollados diversas versiones de la plataforma *JAVA* persiguiendo el hecho de cubrir las diferentes necesidades de los usuarios.

Según Rojas & Lucas (2003), “el paquete JAVA 2 lo podemos dividir en 3 ediciones distintas. J2SE (*JAVA Standard Edition*) orientada al desarrollo de aplicaciones independientes de la plataforma, J2EE (*JAVA Enterprise Edition*) orientada al entorno empresarial y J2ME (*JAVA Micro Edition*) orientada a dispositivos con capacidades restringidas”.

La aplicación móvil desarrollada en el proyecto al cual refiere este informe, se enmarca bajo el entorno J2ME. La misma es utilizada no sólo en dispositivos móviles

de comunicación (PDAs[2], Tablets, Celulares), sino que la misma es usada actualmente en dispositivos electrónicos con capacidades computacionales y graficas reducida tales como electrodomésticos inteligentes. "*Esta edición tiene unos componentes básicos que la diferencian de las otras versiones, como el uso de una máquina virtual denominada KVM (Kilo Virtual Machine, debido a que requiere sólo unos pocos Kilobytes de memoria para funcionar) en vez del uso de la JVM clásica y la inclusión de un pequeño y rápido recolector de basura*" [3]

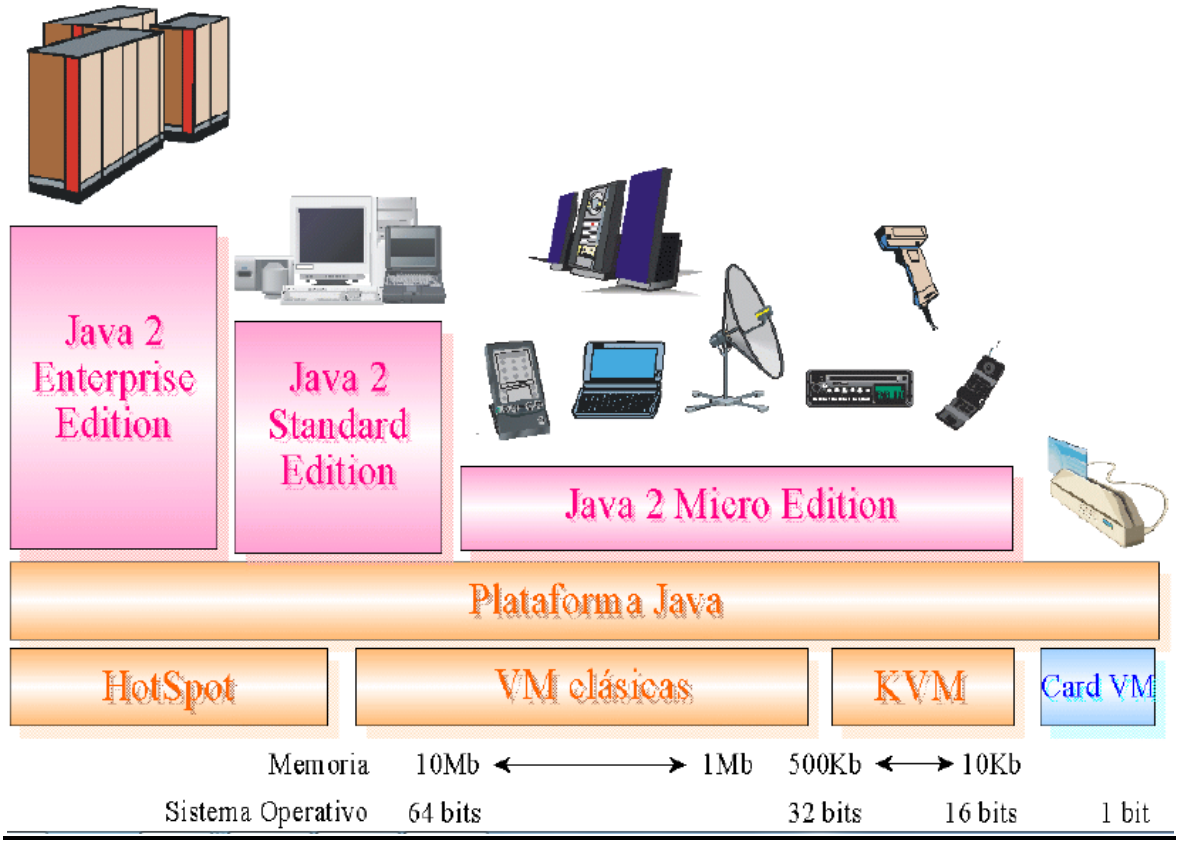

Figura 4. Arquitectura de la plataforma JAVA 2 de Sun.

Fuente: Rojas Sergio, Díaz Lucas. JAVA a Tope: J2ME. 2003. ISBN: 84-688-4704-6

[2] PDA (del inglés *personal digital assistant)* o asistente digital personal, es una computadora de mano originalmente diseñado como agenda electrónica (calendario, lista de contactos, bloc de notas y recordatorios) con un sistema de reconocimiento de escritura.

[3] Tomado de Rojas Sergio, Díaz Lucas. *JAVA a Tope: J2ME*. 2003. ISBN: 84-688-4704-6. Pág. 3

2.4 *Servlet.*

El *Servlet* es una clase *JAVA* que se ejecuta en un servidor de aplicaciones, el cual, cumple funciones de ofrecer respuestas a peticiones específicas hechas por dispositivos que actúan como clientes de un determinado servidor. El mismo puede ser comparado con programas *Common Gateway* Interface ó *CGI* (aplicaciones escritas mayormente en *PERL*, que ejecutándose en un servidor *web* también da respuestas a peticiones hechas por clientes).

Básicamente un *Servlet* recibe una petición de un usuario y luego retorna un resultado relacionado con la petición hecha en primer lugar. Una aplicación frecuente que se la da a estos, es la búsqueda de diversos parámetros en una base de datos.

Rodriguez, García de Jalón, & Imaz (1999) afirman la existencia de distintos tipos de *Servlets*, no sólo limitados a servidores Web y al protocolo *HTTP* (*HyperText Transfer Protocol*), es decir, es probable diseñar *Servlets* que se ejecuten en servidores de correo o FTP[4], sin embargo en la mayoría de los casos, únicamente se utilizan *Servlets* en servidores HTTP.

Centrándose en los *Servlets HTTP*, el cliente del *Servlet* será un navegador Web que realiza una petición utilizando para ello una cabecera de petición HTTP (o cualquier aplicación que use dicho protocolo para realizar las demandas). El *Servlet* procesará la petición y realizará las operaciones pertinentes, estas operaciones pueden ser todo lo variadas que precisemos.

Normalmente el *Servlet* devolverá una respuesta en formato HTML al navegador que le realizó la petición, esta respuesta se generará de forma dinámica. En los *Servlets* no existe una clara distinción entre el código HTML (contenido estático) y el código JAVA (lógica del programa), el contenido HTML que se envía al cliente se

[4] FTP (sigla en inglés de File Transfer Protocol - Protocolo de Transferencia de Archivos) es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP (Transmission Control Protocol)

suele encontrar como cadenas de texto (objetos de la clase JAVA.lang.*String*) que incluyen instrucciones out.println("<h1>CódigoHTML</h1>"). Esto ocurría también con los predecesores de los *Servlets*, es decir, los scripts CGI (*Common Gateway Interface*).

El API *Servlet* 2.2 posee una serie de clases e interfaces que nos permiten interactuar con las peticiones y respuestas del protocolo HTTP. De esto se deduce que para entender los *Servlets* y poder realizarlos de manera correcta deberemos comprender también el protocolo HTTP (*HyperText Transfer Protocol*).

Los *Servlets HTTP* implementan la especificación *Servlet* 2.2 pero adaptándola al entorno de los servidores Web e INTERNET.

La especificación que definen los *Servlets* en su versión 2.2 la encontramos implementada mediante dos paquetes que podemos encontrar formando parte de la plataforma *JAVA 2 Enterprise Edition*. El paquete *JAVAx.servlet*, define el marco básico que se corresponde con los *Servlets* genéricos y el paquete *JAVAx.servlet.http* contiene las extensiones que se realizan a los *Servlets* genéricos necesarias para *los Servlets HTTP*, que como ya hemos dicho son los *Servlets* que vamos a tratar en el texto.

Los *Servlets* pueden utilizar todos los *API's* que ofrece el lenguaje JAVA, además tienen las mismas características que define JAVA, y por lo tanto son una solución idónea para el desarrollo de aplicaciones *Web* de forma independiente del servidor *Web* y del sistema operativo en el que se encuentren, esto es así gracias a la característica que ofrece JAVA de independencia de la plataforma.

Los *servlets* genéricos se encuentran ubicados en el paquete *JAVAx.servlet*, y los *Servlets HTTP* se encuentran en un sub-paquete del paquete anterior, este paquete se conoce como *JAVAx.servlet.http.*

La dependencia de los *Servlets* con el protocolo *HTTP* se hace patente en las distintas clases e interfaces que se encuentran en el paquete *JAVAx.servlet.http*. Así

por ejemplo el interfaz *HttpServletRequest* representa la petición que se ha realizado a un *Servlet*, es decir, una cabecera de petición del protocolo *HTTP*, y nos permite obtener información variada de la misma, como se puede comprobar el nombre del interfaz es bastante intuitivo. De esta misma forma existe un interfaz llamado *JAVAx.servlet.http.HttpServletResponse*, que representa una cabecera de respuesta del protocolo *HTTP* y que nos ofrece un conjunto de métodos que nos permiten manipular y configurar de forma precisa la respuesta que se le va enviar al usuario (Rodriguez, García de Jalón, & Imaz, 1999).

Para poder ejecutar *Servlets* en un servidor Web, el servidor Web debe disponer motor o contenedor de *Servlets*, en algunos casos el contendor o motor es un añadido que se instala como un complemento al servidor Web, y en otros casos el propio servidor Web es también un motor de *Servlets*.

Por otro lado los *Servlets* pueden ser categorizados por el tipo de tareas para las cuales se diseñan los mismos. A continuación se presenta la estructura que define los mismos tomando en cuenta el orden lógico en el que se realizan las tareas:

- Leer los datos enviados por el usuario: normalmente estos datos se indican a través de formularios *HTML*[5] que se encuentran en páginas Web. Aunque esta información también puede provenir de otras herramientas *HTTP*.
- Buscar otra información sobre la petición que se encuentra incluida en la petición *HTTP*: esta información incluye detalles tales como las capacidades y características del navegador, cookies, el nombre de la máquina del cliente, etc.

[5] HTML, siglas de *Hyper Text Markup Language* (Lenguaje de Marcado de Hipertexto), es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

- Generar los resultados: este proceso puede requerir acceder a una base de datos utilizando *JDBC*[6], utilizar un componente JAVABean, o generar la respuesta de manera directa.

- Formatear los resultados en un documento: en la mayoría de los casos implica incluir los resultados en una página HTML.

- Asignar los parámetros apropiados de la respuesta HTTP: esto implica indicar al navegador el tipo de documento que se le envía (por ejemplo HTML), asignar valores a las cookies, y otras tareas.

- Enviar el documento al cliente: el documento se puede enviar en formato de texto (HTML), formato binario (imágenes GIF[7]), o incluso en formato comprimido como un fichero ZIP[8].

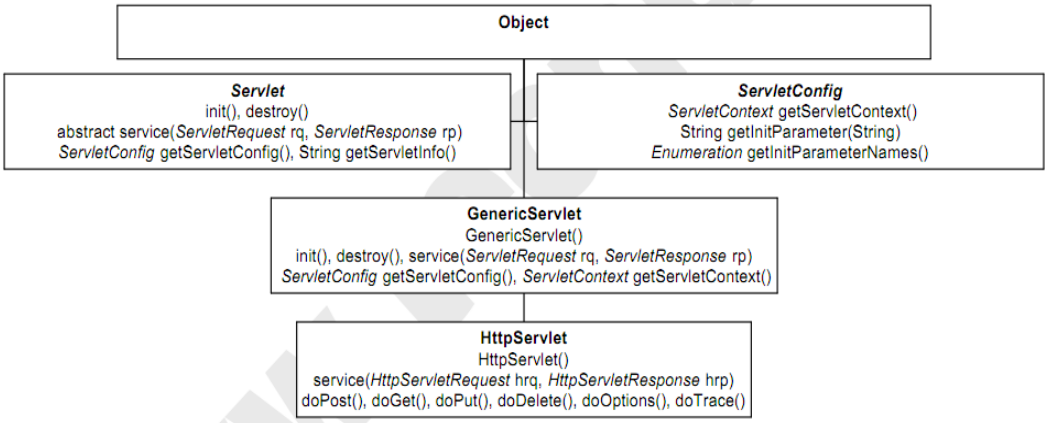

Figura 5. Jerarquía y Métodos de las principales clases para crear Servlets.

Fuente: Javier García de Jalón, José Ignacio Rodríguez, Aitor Imaz. Aprenda Servlets de JAVA Como si Estuviera en Primero. San Sebastián, España. Abril 1999. Pág. 16.

[6] API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación JAVA, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.

[7] Formato gráfico utilizado ampliamente en la *World Wide Web* (WWW), tanto para imágenes como para animaciones.

[8] Formato de almacenamiento sin pérdida, utilizado ampliamente para la compresión de datos como documentos, imágenes o programas.

2.5 Servidor de Aplicaciones

López Franco (2001) define un servidor de aplicaciones como un programa caracterizado por poseer un entorno de ejecución compatible con un determinado lenguaje de programación (como *JAVA*), que a la vez tiene la capacidad de recibir peticiones HTTP, y ejecutar rutinas orientadas a la realización de diversas tareas para generar una respuesta bajo este mismo protocolo, enviándolas al cliente responsable de la petición. Las aplicaciones para las cuales se usan este tipo de servidores se enmarcan bajo la filosofía de la lógica del negocio[9]. Actualmente, el desarrollo de aplicaciones web ha requerido el uso de estos ya que son servidores de alto rendimiento que responden a necesidad de sitios web con gran utilización de recursos (una gran afluencia de visitas, movimientos de datos, transacciones o búsquedas en bases de datos). Generalmente dichos servidores son extensiones que desarrollan los fabricantes de servidores web, los cuales se caracterizan por tener una total compatibilidad en cuanto a la comunicación con los servidores de aplicaciones a través de controladores (*drivers*) desarrollados por los fabricantes. Cabe destacar que la información que viaja de ida y vuelta entre un servidor de aplicaciones y sus clientes no está restringida a simplemente *HTML*. En lugar de eso, la información es lógica de programa. Ya que la lógica toma forma de datos, llamadas a métodos y HTML no estático, el cliente puede emplear la lógica de negocio expuesta a conveniencia.

Este tipo de programas están definidos dentro del estándar J2EE, hoy en día es el más utilizado debido a que se caracteriza por el desarrollo de aplicaciones web de una manera sencilla y eficiente. Las aplicaciones desarrolladas bajo este estándar, pueden ser ejecutadas en cualquier servidor que cumplas con las características de dicho estándar. A continuación se hace referencia a la Arquitectura J2EE en la Figura 6.

[9] Se define como la estructura en la cual reside el servidor de aplicaciones y el conjunto de programas a los cuales proporciona soporte.

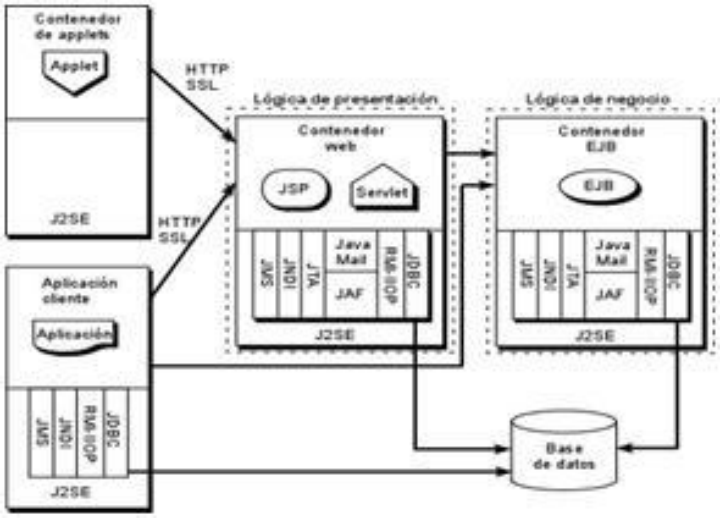

Figura 6. . Arquitectura J2EE.

Fuente: www.jtech.ua.es

En la figura anterior se pueden observar las diversas etapas o estructuras J2SE que conforman la arquitectura J2EE, las cuales se detallan a continuación:

- **Contenedor de Applets:** representa el cliente (aplicaciones móviles, aplicaciones web, navegadores o cualquier programa que realice las peticiones al servidor). El mismo interactúa con el contenedor web haciendo uso de *HTTP*. Recibe páginas *HTML* o *XML* y puede ejecutar código *JAVAScript*.

- **Aplicación cliente:** Son clientes que no se ejecutan dentro de un navegador y pueden utilizar cualquier tecnología para comunicarse con el contenedor web o directamente con la base de datos.

- **Contenedor web:** se le llama usualmente servidor web, y representa la interfaz del servidor de aplicaciones, en el cual se encuentran los *Servlets* y archivos

jsp[10], los cuales poseen el código de ejecución para prestar el servicio web a los clientes dependiendo del tipo de petición que realicen. Este contenedor se vale de la utilización de HTTP y TLS[11] (*Transport Layer Security*) para comunicarse con el contenedor de *applets*.

- **Servidor de aplicaciones:** es el encargado de proporcionar los servicios relacionados con las aplicaciones web que se encuentran en el servidor, es decir, es la estructura a la cual se le delegan las funciones de ejecutar los *servlets*.

La integración del servidor de aplicaciones al servicio web redefine el modelo de capas de un servidor web.

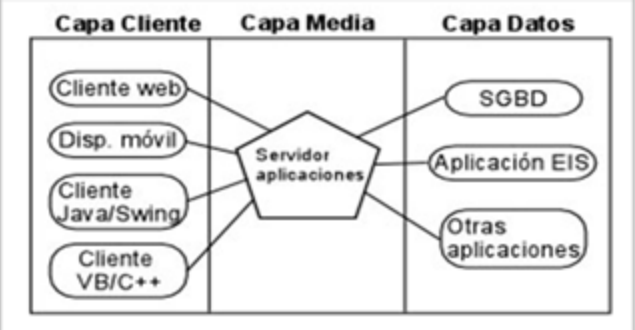

Figura 7. Arquitectura de 3 capas utilizando el servidor de aplicaciones.

Fuente: www.jtech.ua.es

(Esteban, 2000) Expone en su obra que en general, se puede afirmar que las características principales que definen un servidor de aplicaciones, diferenciándole de otros tipos de servidores son:

[10] *JAVAServer Pages* (JSP) es una tecnología *JAVA* desarrollada por SUN MICROSYSTEMS que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo. Además permiten la ejecución del código *JAVA* mediante scripts.

[11] Protocolo criptográfico que proporciona comunicaciones seguras por una red, comúnmente INTERNET.

- **Generación de HTML:** debe incorporar generación dinámica de contenido (HTML, XHTML[12], XML, etc.), para enviar al cliente.

- **Trabajo con bases de datos:** existirán objetos que faciliten el *acceso a bases de datos*, ocupándose de gestionar las conexiones y proporcionando un acceso uniforme. Otros objetos se encargarán de la *gestión de transacciones* englobando diversas sentencias y ocupándose de los *commit* o *rollback*.

- **Funcionamiento multiproceso o multihilo:** el servidor es el responsable de tener funcionando un número de hilos o procesos que atiendan a distintas peticiones.

- **Sesiones:** HTTP es un protocolo sin estados. Un servidor de aplicaciones provee de persistencia a los datos del usuario mediante objetos de sesión (*session*). Elimina la necesidad de incluir código en las aplicaciones para diferenciar las peticiones de distintos usuarios.

- **Lógica de negocio:** la lógica de negocio propia de cada aplicación debe poder ser encapsulada en componentes. A cada uno de ellos se le podrán asignar mecanismos propios de seguridad, gestión de transacciones.

- **Seguridad:** debe poseer características de seguridad que den soporte a aplicaciones seguras. Los clientes deben autentificarse contra al servidor, y este es el responsable de darles acceso a sus diferentes componentes, como puede ser una base de datos. La mayoría de servidores disponen de un mecanismo para incorporar nuevos usuarios y grupos. El control de a que partes del servidor puede acceder un

[12] Acrónimo en inglés de *eXtensible Hypertext Markup Language* (lenguaje extensible de marcado de hipertexto). Lenguaje de marcado pensado para sustituir a HTML como estándar para las páginas *web.*

usuario puede ser controlado por diversos métodos, por ejemplo en un directorio LDAP (*Lightweight Directory Access Protocol*).

- **Balanceo de carga:** trabajando sobre un *cluster* de servidores, puede enviar las peticiones a diferentes equipos en función de la carga y la disponibilidad. Este balanceo es la base para implementar sistemas tolerantes a fallos o herramientas para la monitorización centralizada de todos los equipos del *cluster*.

2.6 Plataforma de Desarrollo *BLACKBERRY*

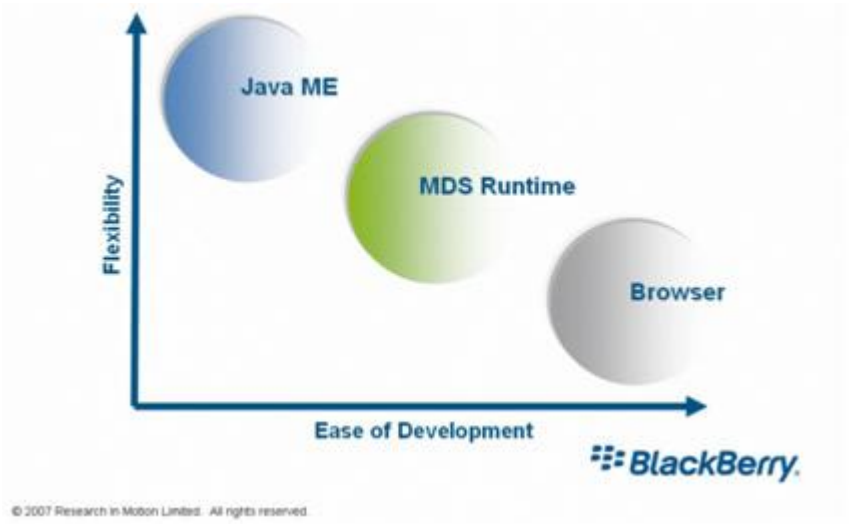

Figura 8. Gráfico comparativo entre los enfoques de programación BLACKBERRY.

Fuente: BLACKBERRY.devcite.com

La figura anterior representa los diferentes enfoques usados para el desarrollo en la plataforma, cada opción cuenta con características específicas que determinan sus ventajas o desventajas dependiendo del contexto en donde se apliquen. En la figura se toman en cuenta dos variables las cuales son: Flexibilidad y Facilidad para el desarrollo de aplicaciones (Riveros).

En primer lugar tenemos *JAVA ME*[13], con la cual se pueden implementar desarrollos más flexibles ya que permite personalizar en gran medida las aplicaciones desarrolladas. Por otro lado se tiene el desarrollo vía *Browser*, donde se hace más sencillo el desarrollo a causa de que se cuenta con un gran número de estándares para diversas funcionalidades propias de las aplicaciones, sin embargo cuenta con algunas limitaciones en cuanto a la presentación de algunas páginas y reproducción de elementos multimedia en páginas *web*. Además se tiene el enfoque *MDS Runtime*[14], el cual funge como conexión entre los dos enfoques explicados anteriormente, ocupando características de los mismos para realizar desarrollos orientados a servicios *web* (*Webservices*). (Riveros, 2007).

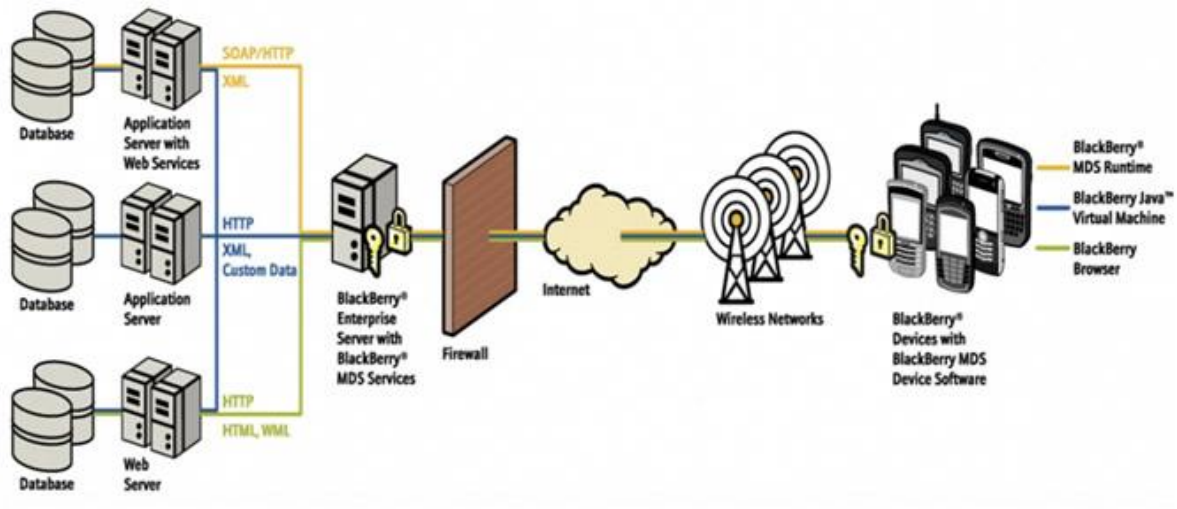

Figura 9. Arquitectura de la Plataforma BLACKBERRY.

Fuente: BLACKBERRY.devcite.com

Todas las opciones de desarrollo en BLACKBERRY ocupan la misma arquitectura para la comunicación. Como se observa en la Figura 4, la comunicación va desde la parte derecha con los dispositivos hacia la parte izquierda donde se

[13] Distribución del lenguaje *JAVA,* orientado al desarrollo de aplicaciones en dispositivos móviles.

[14] Acrónimo de *Mobile Data System*. Es la plataforma responsable de que la aplicación tenga acceso a los recursos ubicados detrás del firewall de una empresa sin software adicional VPN. Además *BlackBerry MDS* proporciona *proxies HTTP y TCP/IP* para las aplicaciones.

desarrollan las aplicaciones. Luego se observa las redes inalámbricas o *Wireless Networks* (donde existen diferentes estándares entre ellos EDGE o HDSP), hacia la infraestructura *BLACKBERRY*, la cual conectará con el *BLACKBERRY Enterprise Server* con BLACKBERRY *MDS Services* donde lo primero que esta es el *Firewall* que generará la conectividad entre el dispositivo dentro de la nube de INTERNET hacia la infraestructura *BLACKBERRY* **(**Riveros**)**.

Conforme a lo anterior, se puede afirmar que la información va desde el dispositivo a la infraestructura, de forma tal que se va automáticamente por el correspondiente canal abierto del BLACKBERRY Enterprise Server, así dependiendo de la información o el tipo de flujo de información provista por el dispositivo, tomará el camino más apropiado para que esta sea entregada a las diferentes tecnologías que toman parte de esta arquitectura, tales como: Servidores de Aplicaciones (*Application Servers)*, Servicios *Web (Webservices)* o la combinación de ambos.

La línea naranja en la ilustración, representa la BLACKBERRY MDS *Runtime*, que conecta a la red inalámbrica hacia el BLACKBERRY ENTERPRISE Server y luego hacia el *MDS server* corriendo ahí, con lo que usamos *SOAP*[15] y *Webservices*, para conectar el *Application Server* vía *Webservice* o HTTP.

De la misma forma con JAVA, pasa por las redes Wireless hacia INTERNET y luego al *BLACKBERRY MDS Services*, el cual deriva hacia los tres tipos de tecnologías o *Back-ends* que describimos anteriormente, siendo este enfoque muy flexible ya que existe comunicación ya sea por HTTP, XML o algún protocolo propio que se desarrolle.

BLACKBERRY Browser en cambio, va desde el *BLACKBERRY Enterprise Server* conectando con *BLACKBERRY MDS* server y luego vía *HTTP* conecta al *Webserver* en el *Back-end.*

[15] Siglas de *Simple Object Access Protocol*. Protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

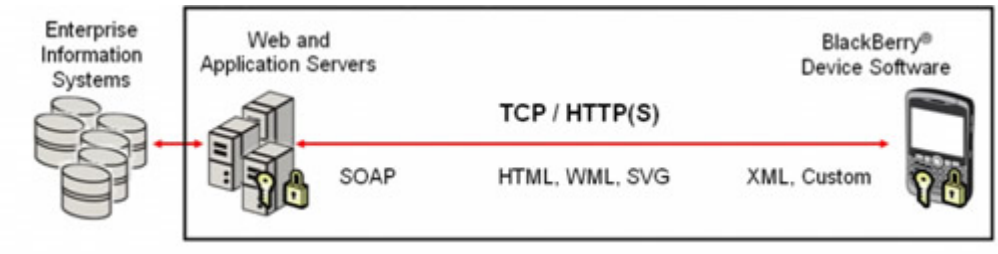

Figura 10. Arquitectura BLACKBERRY vista de la perspectiva de la aplicación.

Fuente: BLACKBERRY.devcite.com

El desarrollador no está obligado a comprender el funcionamiento a fondo de la arquitectura expuesta anteriormente, es decir, la manera en cómo la información está siendo manejada y redirigida a través de la redes o de acuerdo a qué tipo de señal se está comunicando, esto es automático. Asimismo se puede decir que esta arquitectura funciona como una VPN (*Virtual Private Network*) permanente. Para esto *BLACKBERRY MDS* provee de Seguridad *End-to-End*, afuera de esta caja negra, como también provee de la administración de la conectividad inalámbrica, Protocolos de Interfaces Estándar como HTML, WML, SVG, etc. Y provee de independencia para el dispositivo como para la red, ya que no importa si es utilizado el estándar EDGE, HDSP, etc., todo es automático y transparente.

2.7 Sistema Administrador de Bases de Datos Relacionales (RDBMS)

Es definido como un programa que ofrece una interfaz entre una base de datos y el usuario o administrador de la misma. De esta forma se hace más sencillo el manejo de una base de datos que almacena valores o parámetros de importancia para un sistema específico. Además cuentan de diversas características que orientan a los usuarios a hacer el uso de la misma. La principal seria la abstracción de la información, ya que para el usuario no es visible la cantidad de archivos que normalmente conforman una base de datos, permitiéndole al usuario enfocarse solo en los datos que conforman la misma.

Por otro lado este sistema de gestión ofrece independencia de datos, ya que se puede modificar la estructura de la base de datos sin alterar la configuración o el

funcionamiento de las aplicaciones que dependen o están asociadas a la base de datos. Además ofrecen la posibilidad de personalizarlas a gusto del administrador, ya que se pueden condicionar entradas de datos a la base de datos según crea conveniente el diseñador de la misma. También ofrecen un tiempo de respuesta mínimo (casi inmediato) al modificar la estructura de la base datos. De igual forma garantiza la seguridad para los datos que conforman la base de datos mediante la creación de permisos y claves para diferentes usuarios definidos dentro de la estructura de la misma.

Entre los *RDBMS* más conocidos se pueden mencionar: *PostgreSQL, MySQL* y *SQL Server.*

2.8 Protocolo TCP/IP.

Según (Forouzan, 2007), es un protocolo con una estructura jerárquica, el cual se compone de un conjunto de módulos interactivos, los cuales ofrecen una funcionalidad especifica; sin embargo, dichos módulos no trabajan de manera independiente. Los niveles de la familia de protocolos TCP/IP contienen protocolos relativamente independientes que pueden ser mezcla o coincidir dependiendo de las necesidades del sistema. El término jerárquico referente a la estructura del protocolo se debe a que cada protocolo del nivel superior es soportado por uno o más protocolos del nivel inferior. TCP/IP tiene cuatro capas de abstracción según se define en el *RFC 1122*, los cuales se definen a continuación:

2.8.1 Nivel Físico y de Enlace de Datos

En este nivel, TCP/IP no define ningún protocolo específico. Soporta todos los protocolos estándar y propietarios. El *software* TCP/IP de nivel inferior consta de una capa de interfaz de red responsable de aceptar los datagramas IP y transmitirlos

hacia una red específica. Una interfaz de red puede consistir en un dispositivo controlador (por ejemplo, cuando la red es una red de área local a la que las máquinas están conectadas directamente) o un complejo subsistema que utiliza un protocolo de enlace de datos propios (por ejemplo, cuando la red consiste de conmutadores de paquetes que se comunican con anfitriones utilizando HDLC[16]).

2.8.2 Nivel de Red

También conocido como nivel de interconexión, soporta el protocolo de interconexión IP, a su vez, utiliza cuatro protocolos de soporte: ARP, RARP, ICMP e IGMP. En esencia, esta capa maneja la comunicación de una máquina a otra. Ésta acepta una solicitud para enviar un paquete desde la capa de transporte, junto con una identificación de la máquina, hacia la que se debe enviar el paquete. La capa INTERNET también maneja la entrada de datagramas, verifica su validez y utiliza un algoritmo de enrutamiento para decidir si el datagrama debe procesarse de manera local o debe ser transmitido. Para el caso de los datagramas direccionados hacia la máquina local, el software de la capa de red borra el encabezado del datagrama y selecciona, de entre varios protocolos de transporte, un protocolo con el que manejará el paquete. Por último, la capa o nivel de red envía los mensajes ICMP de error y control necesarios y maneja todos los mensajes ICMP entrantes.

2.8.2.1 Protocolo de INTERNET (IP o *INTERNET Protocol*)

Es el mecanismo utilizado para la transmisión en TCP/IP. Es un protocolo no fiable y no orientado a conexión. El mismo no ofrece ninguna comprobación ni seguimiento de errores. IP asume la no fiabilidad de los niveles inferiores y hace lo mejor que puede para conseguir una transmisión a su destino pero sin garantías. Transporta los datos en paquetes llamados datagramas, cada uno de los cuales se transporta de forma independiente. Los datagramas pueden viajar por diferentes rutas

[16] *High-level Data Link Control* (Control de Enlace Síncrono de Datos).

y pueden llegar fuera de secuencia o duplicados. IP no sigue la pista de las rutas y no tiene forma de reordenar los datagramas una vez que llegan al destino. Sin embargo las limitaciones de este protocolo no se deben considerar como debilidad ya que ofrece funciones de transmisión básicas y deja libertad al usuario para añadir solo aquellas funcionalidades necesarias para una aplicación determinada y por tanto se ofrece la máxima flexibilidad.

2.8.3. Nivel de Transporte

Este nivel fue representado originalmente por dos protocolos: UDP y TCP. Ip es un protocolo *host* a *host*, lo que significa que puede entregar un paquete desde un dispositivo físico a otro. UDP y TCP son protocolos del nivel de transporte encargados de la entrega de mensajes desde un proceso (programa en ejecución) a otro proceso. Además se desarrollo SCTP, este protocolo se implementa con la finalidad de cumplir con los requerimientos de otras aplicaciones (Forouzan, 2007).

2.8.3.1 Protocolo de Datagramas de Usuario (UDP)

Es un protocolo proceso a proceso que añade solo direcciones de puertos, control de errores mediante sumas de comprobación e información sobre la longitud de los datos del nivel superior. Este protocolo es el más sencillo de los dos protocolos estándar en TCP/IP.

2.8.3.2 Protocolo de Control de Transmisión (TCP)

Este protocolo ofrece servicios completos de nivel de transporte a las aplicaciones. TCP es un protocolo de flujos fiable, es decir, el mismo es un protocolo orientado a conexión: se debe establecer una conexión entre los dos extremos de la transmisión antes de que se puedan transmitir datos.

En el extremo emisor de cada transmisión, TCP divide un flujo de datos en unidades más pequeñas denominadas segmentos. Cada segmento incluye un número

de secuencia para su reordenación en el receptor, junto con un número de confirmación para los segmentos recibidos. Los segmentos se transportan a través de datagramas IP. En el extremo receptor, TCP recibe cada datagrama y reordena la transmisión de acuerdo a los números de secuencia.

2.8.3.3 Protocolo de Transmisión de Control de Flujos (SCTP)

Este protocolo ofrece soporte para aplicaciones como voz sobre IP. Este protocolo de transporte combina las mejores características presentes en UDP y TCP.

2.8.4 Nivel de Aplicación

Es el nivel más alto, los usuarios llaman a una aplicación que acceda servicios disponibles a través de la red de redes TCP/IP. Una aplicación interactúa con uno de los protocolos de nivel de transporte para enviar o recibir datos. Cada programa de aplicación selecciona el tipo de transporte necesario, el cual puede ser una secuencia de mensajes individuales o un flujo continuo de octetos. El programa de aplicación pasa los datos en la forma requerida hacia el nivel de transporte para su entrega.

2.8.5 Seguridad de los Datos

La seguridad de los datos es esencial, sobre todo al momento enviar toda esa información a través de la red. Muchas veces las expectativas de seguridad ofrecidas pueden verse frustradas por factores intencionales como algún intruso que logra entrar en el sistema o interviene en los medios físicos de la red para captar la información.

Para poder ofrecer garantías en cuanto a confidencialidad es necesario que toda la información con la que se trabaje se encuentre cifrada, para ello existen diferentes algoritmos de encriptación, entre los que están:

- DES (*Data Encryption Standard*): Es un algoritmo de cifrado cuyos orígenes se remontan a 1970, su operación se basa en tomar un texto en claro de una

longitud fija de bits y transformarlo mediante una serie de operaciones en otro texto cifrado de la misma longitud. El algoritmo DES utiliza una clave criptográfica de 56 bits para modificar la transformación, de modo que el descifrado solo pueda realizarse por aquellos que conozcan dicha clave. Hoy en día DES se considera inseguro, debido principalmente a que el tamaño de la clave de 56 bits es corto y demuestra debilidades teóricas en su cifrado, por ello actualmente se considera que este algoritmo es seguro en la práctica en su variante Triple DES.

- Triple DES (*Triple Data Encryption Standard*): Cuando se observó que la clave de 56 bits utilizado por DES no era del todo segura, se eligió Triple DES como una forma de agrandar el tamaño de la clave sin necesidad de modificar el algoritmo de cifrado, triplicando el tamaño de la clave efectiva utilizada a 168 bits y logrando que el algoritmo sea inmune a ataques a los que era vulnerable DES. Básicamente Triple DES utiliza tres claves de tipo DES de 56 bits comprimidas en una sola clave de 168 bits, durante su proceso de cifrado, toma los datos y aplica el algoritmo DES tres veces, primero encripta los datos con una de las claves de 56 bits, toma esos datos cifrados y los des encripta utilizando una segunda clave, esto da como resultado un texto doblemente cifrado por haberse utilizado claves diferentes, y finalmente aplica un proceso de encriptado con la tercera clave logrando que los datos estén triplemente cifrados. Para el proceso contrario, el des encriptado de los datos, Triple DES toma estos datos triplemente cifrados y realiza un proceso inverso, es decir, en vez de encriptar-des encriptar-encriptar, esta vez des encripta-encripta-des encripta siempre utilizando las mismas tres claves extraídas de la clave de 168 de Triple DES.
- AES (*Advance Encryption Standard*): Entre los algoritmos de cifrado ya estudiados el AES es el más avanzado de los tres y es uno de los algoritmos

más usados en criptografía simétrica[17]. Al contrario de su predecesor Triple DES, el algoritmo AES puede utilizar una clave secreta de 128, 192 o 256 bits y se basa en un principio de diseño conocido como Red de Sustitución Permutación[18], el número de rondas que realiza el algoritmo AES durante el proceso de sustitución-permutación entre la información sin cifrar y la clave secreta, depende de la longitud de esta última, si la clave es de 128 bits se ejecutan 10 rondas, si es de 192 son 12 rondas y para una clave de 256 bits, que ofrece mayor seguridad, se ejecutan 14 rondas.

2.8.6 Codificación a Base64

Al momento de aplicar algoritmos criptográficos a un texto en claro (sin cifrar), se genera un conjunto de caracteres que representan la información ya cifrada, dicho conjunto de caracteres pueden sufrir cambios o perdidas al momento de ser enviados a través de INTERNET, esto debido a que no todos los componentes de una red necesariamente interpretan cada uno de esos caracteres de la misma manera lo que puede generar errores al momento de intentar descifrar la información.

Para evitar este tipo de problemas se puede implementar la codificación a Base64 que se basa en el uso de los caracteres US-ASCII[19] para codificar cualquier tipo de información utilizando un alfabeto de 64 caracteres imprimibles para representar 6 bits de datos. Los 64 símbolos utilizados son universalmente legibles por lo que al aplicar la codificación Base64 a un texto cifrado por alguno de los algoritmos (DES, Triple DES o AES) se puede evitar el problema de la errónea interpretación de dicha información cuando es enviada a través de la red.

[17] Método criptográfico en el cual se usa una misma clave para cifrar y descifrar mensajes.
[18] Método de cifrado que está compuesto de una serie de sustituciones y permutaciones que se suceden unas a otras.
[19] (*Código Estándar Americano para el Intercambio de Información).* Código de caracteres basado en el alfabeto latino.

CAPITULO III

MARCO METODOLÓGICO.

3.1 Fase de Investigación.

Se procede a analizar y familiarizar con el proyecto, para luego estudiar todo lo relacionado con el lenguaje de programación *JAVA* y el conjunto de herramientas para llevar a cabo el desarrollo de aplicaciones funcionales sobre dispositivos móviles *BLACKBERRY*, usando como fuentes directas de información libros, artículos publicados y foros de desarrolladores especializados con contenido específico referente a los siguientes tópicos:

- Lenguaje de programación usado (*Java*).
- Servidores de Aplicaciones o Contenedores de *Servlets*.
- Bases de Datos y Gestores de las mismas.
- Aplicaciones para dispositivos móviles.
- Ambientes Integrados de Desarrollo tanto para aplicaciones móviles como para aplicaciones *Web* (*IDE*).

Con esta fase se pretende obtener las herramientas necesarias para comenzar con el diseño la aplicación de prueba y elaborar el marco teórico del Trabajo Especial de Grado.

3.2 Fase de estudio y desarrollo

En esta fase, es evaluada la información teórica obtenida. Esta evaluación, se considera necesaria para la comprensión apropiada del funcionamiento de los elementos a escoger, los cuales integraran el entorno de ejecución que se pretende diseñar y emular. Posteriormente, al comprender las características y el funcionamiento de los elementos escogidos, se procede a implementar y configurar las herramientas que se consideran indispensables para garantizar el cumplimiento de los objetivos planteados anteriormente. El diseño concebido para la emulación del ambiente de ejecución de pruebas de la aplicación móvil, estará basado en la configuración de diversos elementos como se describe a continuación:

Para alojar los datos mostrados por la aplicación, se diseñará una base de datos en el gestor escogido (*MySQL*). En la misma estarán incluidas tablas con los datos referentes a las variables supervisadas por la aplicación. De igual forma, se configurará la manera para establecer la comunicación de este gestor para responder a peticiones hechas por el servidor de aplicaciones.

El servidor de aplicaciones establecerá un punto de enlace entre la aplicación móvil a diseñar y el gestor de bases de datos, se plantea configurar un servidor de aplicaciones que reciba y de respuesta a peticiones realizadas por la aplicación móvil diseñada estableciendo conexiones con el gestor de base de datos. La comunicación entre el servidor y el gestor, así como el transporte de datos entre los mismos, estará condicionada por una aplicación web diseñada para ejecutarse en el servidor de aplicaciones. Dicha aplicación web también definirá la comunicación entre el servidor y el simulador en el cual es emulada la aplicación móvil diseñada.

Por otro lado, se plantea la utilización de un IDE para el diseño de la aplicación móvil y la aplicación web, el cual permita realizar las emulaciones de las

mismas en un simulador de dispositivo móvil (en el caso de la aplicación móvil) y en un servidor de aplicaciones (en el caso de la aplicación *web*).

De igual forma se pretende, realizar emulaciones de la aplicación móvil diseñada en diferentes modelos de dispositivos móviles pertenecientes a la firma *BLACKBERRY*. Los modelos referentes a los dispositivos escogidos son los siguientes: 8900, 8520, 9630, 9550 y 9700. Los mismos funcionan bajo *BLACKBERRY* OS5 como sistema operativo nativo. Las emulaciones respectivas a cada modelo, serán realizadas en los simuladores contenidos en el IDE escogido para realizar el diseño de la aplicación móvil. Posteriormente, se evaluará el desempeño de todas las funciones de la aplicación diseñada, teniendo como punto de apoyo para la evaluación del *software*, el cumplimiento de los objetivos señalados en el capítulo I de la presente investigación.

3.3 Fase de Culminación

Por ser ésta la última fase, se realizaron las conclusiones y recomendaciones basadas en los resultados obtenidos producto de las emulaciones en los distintos simuladores escogidos para realizar las pruebas. Además se elaboró el tomo final del Trabajo Especial de Grado, considerando en el aspecto formal las pautas de la UCAB y las Normas APA.

CAPITULO IV

DESARROLLO

En el presente capítulo, se explica de forma detallada las etapas que conforman y definen el funcionamiento de la aplicación móvil diseñada. De igual forma se aborda el tema del funcionamiento y diseño de la aplicación web ejecutada en el servidor de aplicaciones *TOMCAT*.

Para realizar las pruebas del funcionamiento de la aplicación móvil al ejecutarse en el simulador, es necesario configurar un entorno de ejecución el cual está orientado a emular un sistema para establecer una comunicación usando el protocolo HTTP con la aplicación móvil, así como simular el transporte de datos entre la cada uno de los elementos escogidos y configurados. Los elementos que conforman dicho entorno de ejecución son los siguientes:

- Servidor de Aplicaciones (*Apache TOMCAT*).
- Simulador de teléfono inteligente *BLACKBERRY*.
- *Blackberry MDS Service Simulator 4.2.*
- *MySQL Community Server*.

En la Figura 11 se observa la interconexión entre los elementos antes mencionados que definen el entorno de ejecución necesario para emular el funcionamiento de la aplicación móvil.

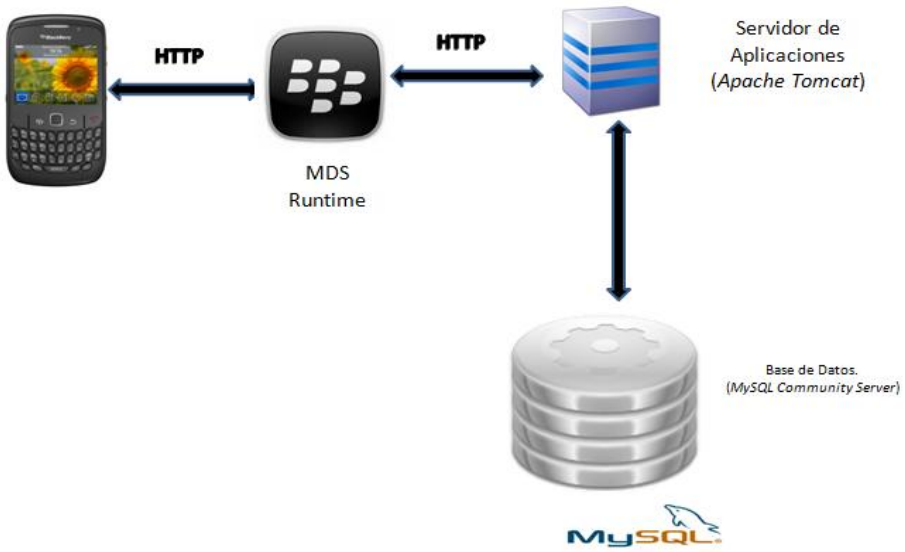

Figura 11. Esquema básico del ambiente de ejecución.

Fuente: Elaborado por los autores

A continuación se explica con detalle las acciones realizadas para llevar a cabo la configuración del entorno referenciado en la Figura 11.

Cabe destacar que cada uno de los elementos que conforman el entorno de ejecución antes mencionado se instalaron y configuraron en un mismo computador con sistema operativo *WINDOWS 7 PROFESSIONAL x86*[20].

Se instaló el servidor de aplicaciones antes mencionado en el computador destinado a ejecutar las emulaciones. El servidor de aplicaciones *TOMCAT* se configuró para recibir y enviar los datos por el puerto 8080. Por otro lado, para que la aplicación pudiese establecer conexiones con el mismo, se consideró pertinente asignar una dirección del tipo *DNS* a la dirección IP en donde se ejecuta el mismo. Como las simulaciones son ejecutadas de manera local, la dirección IP del servidor es 127.0.0.1 ó *localhost,* y la *DNS* de este dominio es "axiserv.com" (el *DNS* antes

[20] Versión más reciente del sistema operativo Microsoft Windows en la línea de sistemas operativos desarrollado por la empresa *Microsoft Corproration*.

mencionado se define en el archivo "*hosts*" localizado en el directorio "C:\Windows\System32\drivers\etc", con la finalidad de que el sistema operativo donde se ejecuta el servidor reconozca la dirección 127.0.0.1 como "axiserv.com").

Por otro lado se instala en el computador, el IDE ó Entorno de Desarrollo Integrado, en el cual se diseña la aplicación móvil y de igual forma, el mismo provee la capacidad de ejecutar la aplicación en los simuladores de los dispositivos móviles escogidos para las emulaciones (Teléfonos Inteligentes *BLACKBERRY* modelos: 8900, 8520, 9630, 9550 y 9700). Además en el IDE se realiza el diseño de la aplicación web la cual se ejecuta en el servidor de aplicaciones, y la misma es responsable de posibilitar la conexión del servidor con el gestor, así como la comunicación con la aplicación móvil. El IDE escogido para esta tarea se conoce como *ECLIPSE*, específicamente la versión "GALILEO". Adicionalmente para desarrollar aplicaciones con capacidad de ser emuladas por simuladores de dispositivos móviles BLACKBERRY, es pertinente instalar el complemento (*Plug-in*) desarrollado por *RIM (Research In Motion)* compatible con este IDE. El Plug-in utilizado se conoce como "*BLACKBERRY eJDE Plug-in for ECLIPSE*", en su versión 1.1.2.201004161203-16.

Otro elemento destacado dentro del sistema diseñado es el *BLACKBERRY MDS* (*Mobile Data System*), el cual es necesario para dar capacidad al simulador de establecer la comunicación con el servidor, así como garantizar el acceso a INTERNET del mismo.

Además se realiza la instalación de un sistema de gestión de bases de datos relacional (*MySQL*) en cual se registran los datos que muestra la aplicación móvil a través de distintas interfaces. En el mismo es declarada una base de datos llamada "axis" en donde se encuentran diversas tablas para alojar información referente a las temperaturas de las cavas, posiciones de los vehículos, usuarios a los cuales se permite el acceso a dichos datos y cantidad de camiones que se encuentran en circulación, así como su respectivo identificador. Este gestor establece conexiones

con el servidor de aplicaciones usando el puerto 3306 para responder a las peticiones hechas por la aplicación móvil. También se debe señalar que el servidor en el cual se ejecuta el gestor tiene como dirección de acceso en la red: *localhost* (127.0.0.1).

Es importante señalar que para habilitar al servidor de aplicaciones en cuanto a realizar conexiones con el gestor de bases de datos relacional, es necesario incluir una librería o paquete *.jar* en el directorio de librerías del mismo que comúnmente se conoce como "conector". Esta librería contiene las clases necesarias para establecer conexiones con el gestor, e intercambiar datos con el mismo con las bases de datos que este maneje, por tanto cuando es creado el proyecto web en el IDE, al especificar el servidor a utilizar, el mismo incorpora al proyecto el paquete de librerías del mismo en el cual se encuentra el "conector" mencionado anteriormente. El "conector" utilizado fue: "*mysql-connector-java-5.1.17*". El mismo se obtuvo al ser descargado desde el sitio web de la empresa que desarrolla el gestor desde el siguiente link: http://dev.mysql.com/downloads/connector/j/.

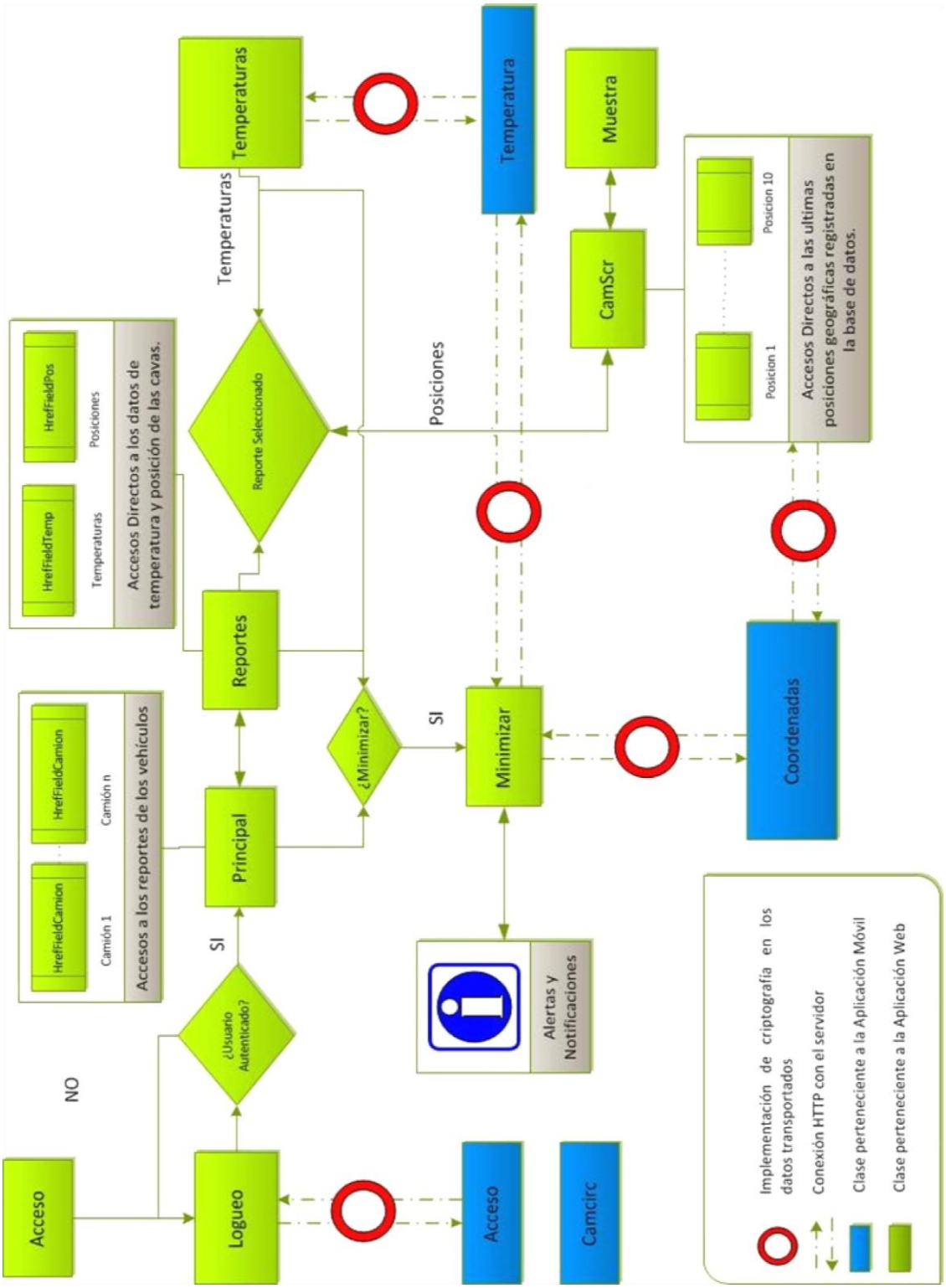

Figura 12. Diagrama de funcionamiento de la Aplicación y el Servidor de Aplicaciones.

Fuente: Elaborado por los autores

En la Figura 12 es representado con detalles la interconexión entre las diferentes clases que definen la aplicación móvil. De igual forma es ilustrado el uso de las clases contenidas en la aplicación web por parte de la aplicación móvil para mostrar los datos recolectados en las diversas etapas de la misma. A continuación es explicado con detalle el funcionamiento de las clases mostradas en la Figura 12 y el propósito para el cual se diseñaron.

4.1 Autenticación del Usuario.

La etapa principal del sistema, se encarga de llevar a cabo el proceso de validación del usuario. La misma consta de dos partes, la rutina ejecutada en el servidor de aplicaciones, y la rutina ejecutada en la aplicación móvil.

Por parte de la aplicación móvil, la rutina se encarga de enviar parámetros de autenticación a un *Servlet* que se ejecuta en el servidor de aplicaciones. El procedimiento encargado de realizar esta tarea tiene como nombre "*login*", y el mismo se encuentra definido dentro de la clase llamada "*Logueo*" (explicada con detalle en el Anexo 15).

El método "*login*", básicamente establece una conexión con el servidor de aplicaciones y envía los parámetros de autenticación (nombre del usuario, contraseña del mismo, y PIN del dispositivo móvil). Para establecer la conexión es usado un objeto del tipo *httpConnection*, y en el mismo se declara la ruta en donde se encuentra el *Servlet* encargado de realizar la comparación de los parámetros suministrados por el usuario con los almacenados en la base de datos.

Este *Servlet* se ubica en la estructura de una aplicación web, se encuentra en el directorio donde se ubican otras clases que realizan diversas tareas relacionadas con el transporte de los datos entre el servidor y la aplicación móvil. Dicha aplicación es llamada "*AxisProy4*". El *Servlet* utilizado es llamado "*Acceso*" y es explicado con detalle en el Anexo 22.

La manera de definir la ruta es la siguiente:

"**url** = http://axiserv.com:8080/Axisproy4/Acceso?nombre="+ usuario+ "&contra=" + contra + "&pin=" + Pin.

En primer lugar se define la dirección en la cual se encuentra el servidor de aplicaciones y el puerto configurado para recibir peticiones y dar respuestas a las mismas (axiserv.com:8080). Luego la aplicación web a la cual se debe accesar (*Axisproy4*) y posteriormente la clase encargada de realizar la verificación de los datos y responder a la petición realizada (*Acceso*) con los datos de autenticación antes mencionados. Es importante señalar que la manera de definir las rutas para accesar a los diferentes servicios ofrecidos por la aplicación web diseñada, es similar a la forma mostrada anteriormente. Solo cambiara la clase accesada dependiendo del servicio a utilizar, es decir, cada clase de la aplicación ofrece un servicio diferente, como se expondrá en los siguientes apartados de este capítulo.

El valor asignado a la variable para establecer la conexión es el siguiente:

(HttpConnection) Connector.open(**url**+";deviceside=false").

Nótese que al parámetro “url” se le añade “deviceside=false”, para indicar que el dispositivo no es un dispositivo real, es decir, como las pruebas se realizaron en un simulador y se hicieron las pruebas de manera local en el computador, es necesario especificar esta condición para poder establecer conexiones HTTP con el servidor. Si se realizaran las pruebas en un dispositivo real, se coloca “deviceside=true” añadiendo otros parámetros para especificar el tipo de red utilizada para la interconexión (wifi, GSM, CDMA).

Al recibir la respuesta del servidor, dependiendo del tipo de la misma, la aplicación se dirige a la pantalla en donde se encuentran botones referentes a cada

camión para obtener la información sobre los mismos (en caso de ser autenticado el usuario) o se redirige a la misma pantalla para volver a introducir los datos de autenticación del usuario.

4.2 Obtención del Numero de Camiones en circulación e Identificadores de los Mismos.

En el caso de que el usuario sea autenticado con éxito, se crea una variable de tipo "*CamCirc"* llamada "cc". "*CamCirc*" es una clase creada durante el desarrollo de la aplicación y no pertenece al conjunto de paquetes que incluye el *JRE* perteneciente al *Plug-in* de *BLACKBERRY* para *ECLIPSE*. Dicha clase se encarga de extraer datos referentes al número de camiones que se encuentran en circulación y de obtener los identificadores pertenecientes a cada camión. La información de identificación referida a cada camión se encuentra alojada en una tabla llamada "*camionescirc*", la cual se encuentra en la base de datos diseñada para el sistema. La clase "*CamCirc*" contenida en la aplicación móvil es descrita en el Anexo 3.

Para la obtención de los identificadores de los camiones en circulación, se ejecuta un *Servlet* llamado "*CamCirc*" ubicado en la aplicación web. El código de esta clase es descrito en el Anexo 24.

Y obtenidos el número de camiones en circulación y sus respectivos identificadores, es creado un objeto para crear una pantalla de la clase "*Principal*". La clase "*Principal*" es una de las pantallas que describen las etapas de la aplicación. Esta pantalla se encarga de mostrar los camiones de circulación y coloca un campo (botón) cuya etiqueta de identificación está relacionada con un identificador de camión. En cuanto al número de campos o botones visibles en la pantalla, dependerá del número de camiones que se encuentren en circulación según la información

suministrada por la base de datos con la cual trabaja el sistema. La clase "*Principal*" se explica con detalle en el Anexo 19.

Por otro lado, además de obtener los identificadores, se recolectan los datos referentes a temperatura y posiciones de las cavas a mostrar cómo se explica en el apartado 4.3.

4.3 Actualización de variables en la aplicación.

La actualización del sistema depende exclusivamente de la clase "*Actualizar*". Para almacenar los datos contenidos en la base de datos, se procede a crear una base de datos en el dispositivo móvil con la clase "*CreaBase*". La misma se describe en el Anexo 5 y está encargada de crear una base de datos en un archivo .db[21] y guardar el mismo en la memoria externa del celular (*microSD*[22]), la función de esta base de datos, es guardar los datos referentes a las temperaturas y posiciones de cada camión que se encuentre en circulación. La creación de la base de datos es llevado a cabo al ejecutar el método "crea" contenido en dicha clase.

Ya creada la base de datos, se ejecuta el método "*act*" contenido en la clase "*Actualizar*" (véase Anexo2).

Posteriormente se ejecuta un ciclo de creación de tablas dentro de la base de datos. Se crea una tabla donde se alojan las ultimas 10 temperaturas de la cava que transporta el camión y se crea otra tabla referente a las ultimas 10 posiciones registradas en la tabla de la base de datos conectada al servidor de aplicaciones.

[21] Extensión del nombre del archivo que identifica al mismo como una base de datos.

[22] Las tarjetas *microSD* o *Transflash* corresponden a un formato de tarjeta de memoria flash desarrollada por SanDisk. Es adoptada por la Asociación de Tarjetas SD bajo el nombre de "*microSD*" en Julio de 2005.

La creación de las tablas de temperatura y posiciones se realizará tantas veces como camiones en circulación se encuentren, es por ellos que el método necesita los parámetros descritos anteriormente. Estas tablas se crean siguiendo las instrucciones del código ilustrado en los métodos "*CreaCoor*" y "*CreaTemp*", los cuales se encuentran definidos en la clase "*CreaTabla*" (véase Anexo 6).

Volviendo al funcionamiento del método "*act*", concluido el ciclo responsable de crear las tablas de temperaturas y coordenadas respectivas por cada camión, se ejecuta uno similar para llenar las tablas creadas con los últimos datos registrados en la base de datos.

Al igual que el ciclo para crear las tablas dentro de la base de datos "*Control*", el ciclo de llenado se ejecuta tantas veces como camiones halla en circulación.

En primer lugar se crea un nuevo objeto perteneciente a la clase "*ExtraeCoor*" (véase Anexo 9) llamado "*exc*". Este objeto recibe como parámetro el identificador del camión del cual se copiaran los datos en la tabla respectiva de posiciones. En resumen, el método "*extrae*" incluido en dicha clase realiza una petición al servidor a través de un *Servlet* contenido en la aplicación web llamado "*Coordenadas*" (descrito en el Anexo 23). Este *Servlet* realiza una búsqueda en la base de datos de la tabla correspondiente al camión del cual se proporciona el identificador y devuelve las últimas 10 posiciones incluidas en la dicha tabla perteneciente a la base de datos.

Para obtener los datos en el dispositivo, se emplea la función "*extrae*" contenido en el objeto "*exc*" perteneciente a la clase "*ExtraeCoor*".

Con respecto a los datos relacionados con la temperatura de cada cava perteneciente a los camiones en circulación, los datos se obtienen con la función "*extrae*" contenido en la clase "*ExtraeTemp*". Tanto la función como la clase se asemejan en gran medida a sus similares en el proceso de extracción de los datos

referentes a las coordenadas que definen la posición de los camiones. La diferencia radica en el *Servlet* a través del cual se realiza la petición hecha al servidor, ya que el mismo accesa a una tabla que contiene los datos con información acerca de la temperatura de las cavas y el tipo de la misma (si es de refrigeración o de congelamiento). Este *Servlet* es llamado "*Temperaturas*" es descrito en el Anexo 26.

Concluidos los procesos para crear la base de datos, crear las tablas para alojar los datos de temperatura y posición por cada camión circulante y extracción de los datos, se procede a realizar el último paso de la fase de actualización, el mismo comprende la inserción de los datos obtenidos de las solicitudes hechas al servidor en las tablas correspondientes. Este proceso se realiza con el objeto "*dat*" referente a la clase "*Datos*". Este objeto contiene dos procedimientos declarados en la clase:

- "*insertacoor*": inserta los datos de posición del vehículo dentro de las tablas "*camionXcoor*" correspondiente a cada uno de los mismos.
- "*insertatemp*": inserta los datos relacionados con la temperatura en las cavas respectivas del vehículo dentro de las tablas "*camionXtemp*".

4.4 Exposición de los Datos en la Pantalla del Dispositivo Móvil.

Anteriormente, en el apartado Autenticación de Usuario se hace mención a la función que cumple la clase que define la pantalla principal de la aplicación luego de la validación de usuario. En la clase "*Principal*" (véase Anexo 19) se define la creación de los botones correspondientes a cada camión haciendo referencia a la clase "*HrefFieldCamion*" (descrita en el Anexo 12).

Dichos botones poseen la propiedad que dado el caso de ser seleccionado con el *trackpad* o *trackball* (dependiendo del dispositivo utilizado), provee al usuario otra pantalla en la cual se muestran dos campos seleccionables los cuales corresponden a los reportes contemplados para la aplicación (posiciones del vehículo y temperaturas de la cava que transporta).

En caso de ser pulsado el campo referente al reporte de temperaturas ("*ht*" en Anexo 20), la aplicación cambia de la pantalla en donde se encuentra el mismo a una pantalla definida por un objeto referente a la clase "*Temperatura*" (véase Anexo 21).

Se definen rangos de temperatura para cada tipo de cava transportada: el rango de temperatura aceptable para una cava de congelamiento es entre -18°C y -25°C. En el caso de una cava de refrigeración, la temperatura es aceptable si se encuentra entre 1°C y 2°C.

La pantalla definida por la clase "*Reportes*" (véase Anexo 20) provee dos campos seleccionables: Uno para accesar a la pantalla "*Temperatura*" (detallada en los párrafos anteriores) y otra para accesar a una pantalla creada a partir de un objeto de la clase "*CamScr*" (véase Anexo 4). Dicha pantalla contendrá un banco de iconos los cuales definen accesos directos a una representación grafica de la posición del camión.

A cada icono se le configuran etiquetas para diferenciarlos entre ellos la cual contiene la hora de la posición registrada.

Cabe destacar que el método para mostrar la posición se realiza de la forma reflejada en el código de la clase (con una imagen estática), dada la imposibilidad de implementar las clases contenidas en las *API´s* propias del *JRE* definido para los mapas y geo-localización.

Adicionalmente tanto las pantallas definidas por las clases "*CamScr*" como la clase "*Temperatura*" posee al final de la pantalla un botón llamado "*Actualizar*" para tener la posibilidad al estar en la pantalla de crear una nueva del mismo tipo con los últimos datos disponibles en la base de datos que se encuentra del lado del servidor de aplicaciones.

Además se le añade una etiqueta en la pantalla posicionada después del botón "*act*" contiene la fecha y hora de la última actualización de la base de datos "*Control*".

Por otro lado, la aplicación ofrece la posibilidad de notificar las alertas al usuario cuando este se encuentra realizando otras tareas en el dispositivo móvil. El procedimiento se ejecuta cuando el usuario selecciona la opción "*Minimizar*" en el menú ubicado en cada una de las pantallas anteriormente detalladas (definidas con las clases "*Principal*", "*Temperatura*", "*CamScr*", "*Muestra*", "*Reportes*"), al seleccionar esta opción se hace un llamado a la clase "*Minimizar*" (véase Anexo 16) que se encargará de actualizar periódicamente los reportes de los camiones en circulación y revisará que todas las temperaturas estén dentro del rango de seguridad establecido, de esta forma si alguna temperatura sale del rango se emite un dialogo de alerta para avisar al usuario.

Se puede destacar que en el menú donde se selecciona la opción de "*Minimizar*" también existe otra opción para salir de aplicación completamente, la misma se define dentro del menú como "*Salir*".

Como último apartado de este capítulo, a continuación se procede a ilustrar los métodos y clases que definen la estructura de encriptación y desencripatación usada en el sistema.

4.5 Encriptado/Des encriptado de los Datos

Para mantener la confidencialidad de los datos al ser enviados a través de la red se pretendió utilizar el algoritmo de encriptación AES para cifrar la información, para ello se decidió desarrollar dos clases que cifraran y descifraran la información tanto en la aplicación móvil como en la aplicación web. Las pruebas para comprobar el funcionamiento de la encriptación consistieron en cifrar la información que la aplicación web extrae de la base de datos y enviarla hasta la aplicación móvil para descifrarla y mostrarla en pantalla. Al momento de ejecutar dichas pruebas surgieron problemas de entendimiento entre la aplicación móvil y la web, ya que al cifrar la información en la aplicación web (mediante el algoritmo AES) y enviarla hasta la móvil, en esta última no se descifraban los datos correctamente y lo mismo sucedía en el proceso inverso. Fue por eso que finalmente se decidió implementar como

mecanismo de encriptación el algoritmo Triple DES que aunque no es tan avanzado y seguro como el AES aún así proporciona un nivel de seguridad lo bastante alto para proteger y mantener la exclusividad de la información al ser enviada a través de la red, además con el Triple DES no surgieron problemas a la hora de realizar las pruebas para comprobar el funcionamiento de la encriptación.

Ya establecido el algoritmo Triple DES como método a utilizar para cifrar los datos que son enviados y recibidos por la aplicación, se desarrollaron las clases "*Crypto*" y "*Ecripta*" (descritas en el Anexo 7 y 25 respectivamente), la primera corre directamente en el simulador del dispositivo móvil y la segunda en el servidor *web*, ambas se encargan de la encriptación y des encriptación de los datos que son enviados desde la aplicación web hasta la móvil y recíprocamente, desde la aplicación móvil hasta la web. Finalmente para un mejor manejo de la data cifrada al ser enviada a través de la red se utilizó la codificación a Base 64, por ende antes que la información sea enviada de la aplicación móvil a la aplicación web y viceversa se lleva a cabo un proceso de Encriptación (con Triple DES) y codificación (con Base 64).

CAPITULO V

RESULTADOS

Este capítulo contiene los eventos o respuestas al ejecutar la aplicación diseñada en los simuladores escogidos para la emulación de la misma: 8900, 8520, 9630, 9550 y 9700. Estas respuestas están relacionadas directamente con la ejecución del código que define cada una de las clases descritas en el capitulo anterior para la aplicación móvil. De igual forma, ya que la aplicación web ("*Axisproy4*"), es parte esencial del sistema diseñado, las clases que integran la misma, hacen posible el cumplimiento de los objetivos para la cual se diseñó la aplicación que se ejecuta en el dispositivo, dada la necesidad del uso del servidor de aplicaciones para manejar las solicitudes hechas por el dispositivo móvil, para luego enviar la respuesta pertinente al mismo.

Tomando el mismo orden descrito en el capitulo anterior para ilustrar el funcionamiento del sistema, se describen a continuación una serie de imágenes referentes al simulador del dispositivo móvil en donde se realizaron las pruebas.

5.1 Pantalla inicial y Menú de acceso:

Se comenzará mostrando la pantalla inicial generada por la clase "*Acceso*".

Figura 13. Pantalla Principal.

Fuente Propia

Como se observa en la figura anterior, la pantalla posee un botón ("Continuar"), el cual al ser seleccionado genera la pantalla descrita en la clase "*Logueo*", la cual como se detalla en el capitulo anterior, cumple funciones de autenticación del usuario de la aplicación. La misma es mostrada en la Figura 14:

Figura 14. Pantalla de ingreso de datos del usuario.

Fuente Propia

Nótese como el PIN del teléfono es expuesto automáticamente en su campo respectivo sin necesidad de que el usuario lo copie en el campo editable. Luego de escribir el nombre y la contraseña correspondiente al usuario se pulsa el botón "Aceptar" para comenzar el proceso de validación con el método "*login*" contenido en la clase "*Logueo*".

Si el nombre, la contraseña y el PIN no coinciden con los datos contenidos en la tabla "*regis*" de la base de datos "*axis*", acontece lo siguiente:

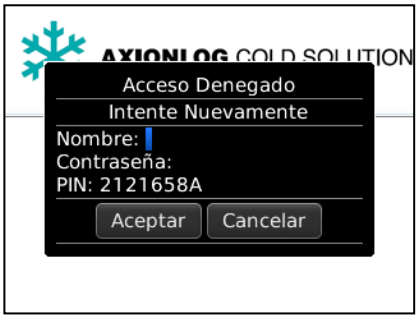

Figura 15. Pantalla que indica un error en los datos personales suministrados.

Fuente Propia

En la Figura 15 se puede observar que se crea un dialogo similar al observado en la Figura 14, solo que tiene como título "Acceso Denegado" para indicar al usuario que alguno de los datos ingresados es erróneo.

Si el usuario es validado con éxito, es mostrada la pantalla generado por la clase "*Principal*".

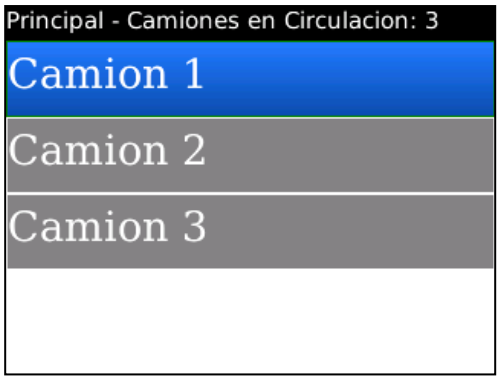

Figura 16. Pantalla que muestra la cantidad de camiones en circulación y sus respectivos vínculos.

Fuente Propia

Se observan en la pantalla anterior los campos seleccionables referentes a los camiones en circulación. En este caso, en la base de datos "*Axis*" se encuentran 3 camiones operativos, por lo que se despliegan 3 campos en la pantalla con los identificadores de los camiones correspondientes en la etiqueta del campo seleccionable.

5.2 Menú de Reportes de los camiones:

Al seleccionar alguno de los camiones de la pantalla Principal (Figura 16), es generada la pantalla que define la clase "*Reportes*" personalizada con los datos referentes al camión seleccionado.

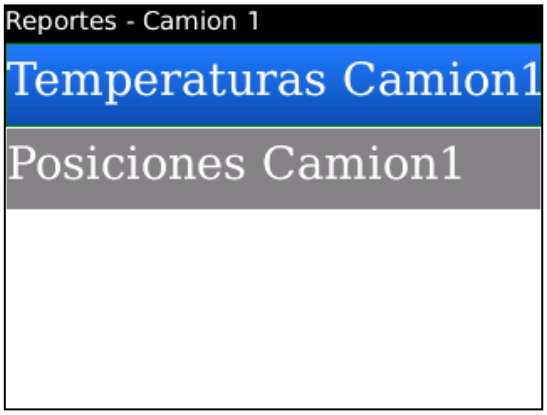

Figura 17. Pantalla con los accesos a los reportes de temperatura y posiciones.

Fuente Propia

Véase como se generan los dos campos seleccionables que corresponden a los accesos a registros de temperatura y posiciones.

5.3 Accediendo a la Temperatura de las cavas de los camiones:

Si es seleccionado "Temperaturas Camion1" se accede a la siguiente pantalla:

Figura 18. Pantalla que muestra el registro de temperaturas (1/3).

Fuente Propia

Como se muestra en la Figura 18, los datos corresponden al camión uno (los datos alojados en la tabla "*camion1temp*"). Los registros de temperatura referentes al valor nominal de la misma son visibles, así como la hora en la cual se registro el evento luego de signo "@". Otro aspecto a destacar es el fondo de la etiqueta donde se muestra la información. El fondo es de color verde dado que las temperaturas se encuentran en el rango de temperaturas considerado aceptable (entre -18°C y -25°C para elementos congelados, entre 1°C y 4°C para elementos en estado refrigerado). Cabe destacar que en la Figura 18 se muestran 4 registros de temperatura y en capítulos anteriores se menciona que la aplicación ofrece las últimas 10 temperaturas registradas. Los 6 registros restantes se muestran en la misma pantalla al navegar por la misma en forma descendente.

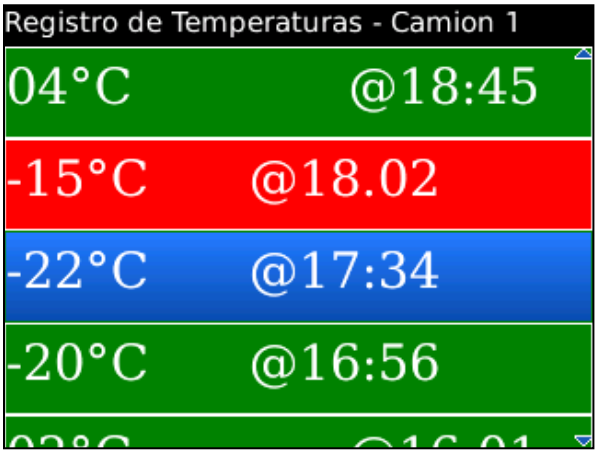

Figura 19. Pantalla que muestra el registro de temperaturas(2/3).

Fuente Propia

Nótese el fondo de la etiqueta referente a "-15°C" en la Figura 19, la cual se encuentra de color rojo debido a no estar en el rango aceptable para una cava de congelado. Al final de la pantalla se ubica una etiqueta que informa la hora y la fecha para la cual se actualizó por última vez la base de datos contenida en la memoria externa del dispositivo (Figura 20). De igual forma se observa el botón que al ser pulsado actualiza la base de datos y genera una pantalla del mismo tipo ("Actualizar").

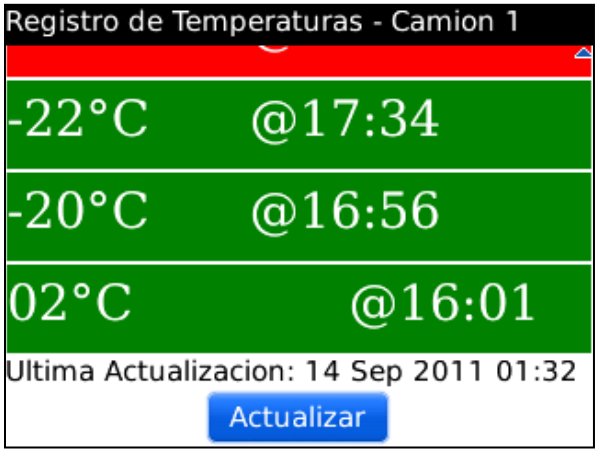

Figura 20. Pantalla que muestra el registro de temperaturas (3/3).

Fuente Propia

5.4 Accediendo a la Posición de los camiones:

Si en el menú mostrado en la Figura 17 (pantalla generada por la clase "*Reportes*") se selecciona el campo con nombre "Posiciones Camion1", se obtiene la pantalla descrita por la clase "*CamScr*" y que se muestra en la siguiente imagen.

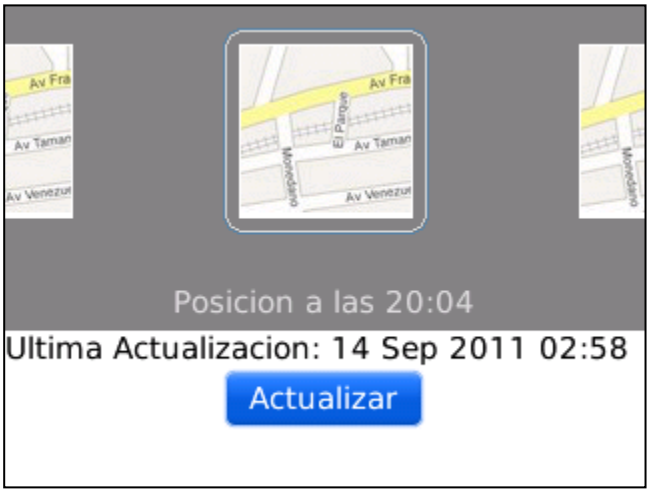

Figura 21. Pantalla de reportes de posiciones.

Fuente Propia

En la Figura 21 se puede apreciar el banco de iconos (fondo oscuro), el cual contiene los accesos para las posiciones registradas. Nótese como al ubicar un icono con el elemento cursor (*trackball* o *trackpad*) del dispositivo, en el mismo se agrega un borde delgado y claro. Además es visible la etiqueta que señala la hora registrada para la posición correspondiente en la parte inferior del icono ("Posicion a las 20:04" en la figura). Del mismo modo se ubican en la parte inferior de la pantalla la etiqueta para informar sobre el momento de la última actualización y un botón que permite realizar la actualización de los registros ubicados en la base de datos "*Control*".

Al ser seleccionado el icono que se encuentra bordeado por la línea delgada clara en la Figura 21, se obtiene la siguiente pantalla:

Figura 22. Ilustración de una posición especifica.

Fuente Propia

En la Figura 22 se ve representada la posición en un mapa geográfico centrado en la posición registrada y se muestran las ubicaciones aledañas a la posición mostrada. De igual forma se señalan en la parte superior los datos correspondientes a la ubicación, reflejando datos como latitud y longitud.

Estando ubicados en la pantalla del mapa geográfico se puede modificar el zoom del mapa accediendo al menú principal (como se muestra en la Figura 23) y seleccionando “Alejar” o “Acercar” según se prefiera, al hacer esto se cargará una nueva imagen del mapa con un acercamiento mayor o menor según la elección del usuario.

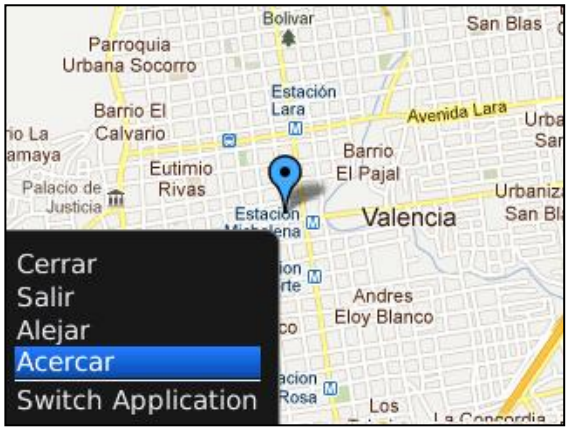

Figura 23. Menú en la pantalla encargada de mostrar las posiciones geográficas.

Fuente Propia

Por ejemplo, si se decide aumentar el zoom de la imagen se selecciona la opción de "Acercar" y la imagen del mapa se cargará nuevamente con un mayor acercamiento como se aprecia en la Figura 24.

Figura 24. Acercando la imagen del mapa.

Fuente Propia

5.5 Modo "background" y Diálogos de Alerta.

Una de las funciones que fueron desarrolladas para la aplicación fue la opción de entrar a un modo "*background*", en el que la aplicación es minimizada más no cerrada completamente, permitiendo al usuario ingresar en otras aplicaciones, revisar sus mensajes, etc. Para acceder a este modo debe seleccionase la opción "Minimizar" del menú principal de la pantalla como se muestra en la Figura 25:

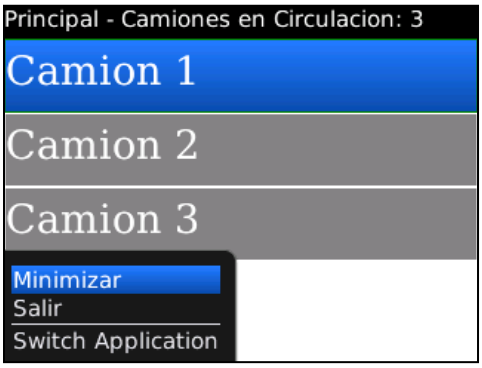

Figura 25. Menú principal que muestra las opciones "Minimizar" y "Salir".

Fuente Propia

La aplicación es cerrada pero el proceso Alertas de la clase "*Minimizar*" realiza sus labores periódicamente para verificar temperaturas fuera del rango aceptable, mientras el usuario puede estar realizando otras actividades dentro del dispositivo.

La clase "*Minimizar*" periódicamente actualizará los registros de temperatura dentro la base de datos "*Control*", si en cierto momento se detectase alguna anomalía en la temperatura de cualquiera de las cavas, la clase "*Minimizar*" mostrará en pantalla diálogos de alerta como el que se muestra en la Figura 26, especificando el camión, la cava, la temperatura y la hora en que fue detectada dicha anomalía:

Figura 26. Alerta generada por la aplicación.

Fuente Propia

Es importante destacar que si el usuario quisiera cerrar por completo la aplicación en vez de minimizarla debe seleccionar en el menú principal (mostrado en la Figura 25) la opción de "Salir" en vez de "Minimizar".

5.6 Pruebas de Encriptado/Des encriptado de Datos.

Para el cifrado de los datos que se envían desde el servidor de aplicaciones hasta el simulador de dispositivo móvil y viceversa, se realizaron pruebas que permitieron observar cómo se mostraban los datos encriptados, con el algoritmo TripleDES y codificados con Base 64.

En la Figura 27 se muestra una captura de pantalla sobre dichas pruebas de cifrado y des cifrado de los datos, para esta prueba se tomó la tabla con las temperaturas de uno de los camiones y se encriptó y codificó para luego realizar el proceso inverso. En la siguiente captura de pantalla se observa primero la "Cadena Encriptada" y luego la "Cadena Desencriptada" arrojándonos los datos correctamente descifrados.

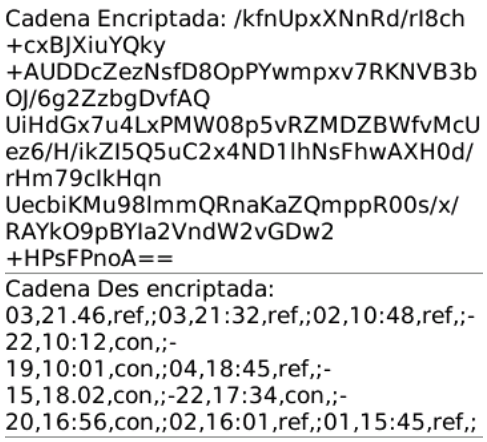

Figura 27. Ejemplo de desencriptación de los datos encriptados provenientes del servidor.

Fuente Propia

Estas pruebas permitieron evaluar el buen funcionamiento de encriptado y codificación de los datos con el algoritmo Triple DES y la codificación a Base 64, antes de que fuesen implementados directamente en la aplicación.

5.7. Funcionamiento de la aplicación en emuladores de dispositivos BLACKBERRY modelos 8900, 8520, 9630, 9550 y 9700.

Como último apartado de este capítulo se considera nececesario mostrar la emulación del funcionamiento de la aplicación en los distintos modelos de simuladores escogidos para realizar las pruebas. Los resultados se ven reflejados en las figuras 28, 29 y 30 como se muestra a continuación:

Figura 28. Emulacion de la aplicación en el dispositivo BLACKBERRY modelo 9700.

Fuente propia.

Figura 29. Emulación de la aplicación en los dispositivos BLACKBERRY 8520 y 8900.

Fuente propia.

Figura 30. Emulacion de la aplicación en los dispositivos BLACKBERRY 9630 y 9550.

Fuente propia.

CAPITULO VI

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

Dado el contenido expuesto en los capítulos anteriores, los objetivos de la investigación reflejada en este Trabajo Especial de Grado se cumplieron a cabalidad en el tiempo esperado según el cronograma establecido, pudiéndose así desarrollar una aplicación enfocada en la supervisión de variables que afectan de manera directa la calidad de los productos alimenticios durante el proceso de distribución de los mismos, garantizando con la aplicación un acceso seguro a la información requerida y aprovechando todas las ventajas que ofrece dicha aplicación al ser compatible con dispositivos móviles *BLACKBERRY*.

Se cumple el objetivo de diseñar una aplicación que posea una interfaz grafica sencilla e intuitiva para ofrecer al usuario los reportes relacionados con las temperaturas de las cavas que transportan los vehículos. De igual forma, las posiciones de los camiones registradas en la base de de datos se muestran de manera explícita en una imagen referente a un mapa geográfico que determina la posición de los mismos. Además la aplicación al ser emulada, y con la implementación de clases especificas en la aplicación móvil, posibilita la generación de alertas o notificaciones visibles en pantalla en caso de detectarse temperaturas de cavas fuera del rango definido como aceptable.

Por otro lado el servidor de aplicaciones escogido para recibir las solicitudes hechas por la aplicación móvil y para dar la correspondiente respuesta a las mismas,

fue configurado de manera correcta para cumplir con dichas funciones, estableciendo el puerto de comunicación con la aplicación, la dirección IP, conectividad con la base de datos y los diversos parámetros descritos en el capítulo 4. Cabe destacar que el mismo representa una parte fundamental en el sistema, dado que sirve de elemento intermedio entre la base de datos y la aplicación móvil.

La base de datos fue diseñada de manera apropiada para almacenar los datos manejados por la aplicación móvil (temperaturas y coordenadas). En la misma se definieron diversas tablas relacionadas con dicha información y fue de vital importancia mantener un orden en cuanto al nombre y contenido de cada una de las tablas, de modo que al desarrollar el respectivo código en *JAVA* para acceder a la Base de Datos no ocurriesen errores al momento de extraer la información requerida de alguna de las tablas. Por otro lado se verificó la conectividad exitosa entre el gestor de base de datos y el servidor de aplicaciones mediante las simulaciones realizadas. Continuando con el tema de las base de datos, resulto eficiente almacenar los datos extraídos a través del servidor web en un gestor de Base de Datos "ligera", como lo es *SQLite*, ya que con el uso del mismo, se almacena de manera esquemática y ordenada la información realizando un símil de la estructura de la base de datos "axis" en la memoria configurada al simulador.

La implementación de seguridad para accesar a los datos que contienen información acerca del posicionamiento de los vehículos y la temperatura respectiva a las cavas que se encuentran en los mismos, se cumple a partir de dos elementos: el primero referido a la implementación, de una interfaz contenida en la aplicación que solicita al usuario de la misma proveer los parámetros que permiten el acceso (nombre y contraseña). Además se decidió incluir el PIN del dispositivo, ya que el mismo es único en cada teléfono comercializado por la empresa responsable de la fabricación de los mismos (*Research In Motion*). Sacando provecho de esta característica, no solo se limita el acceso al usuario de la aplicación (ya que el mismo solo introduce 2 parámetros), sino que es condicionado el acceso al dispositivo en el cual se ejecuta la aplicación móvil. El segundo elemento de seguridad implementado,

es la aplicación de criptografía en los datos transportados entre el servidor y la aplicación móvil, ya que resultaron de gran utilidad los algoritmos Triple DES y Base 64. Por un lado la implementación del algoritmo Triple DES permitió utilizar una clave privada, única para el dispositivo móvil, a partir de la cual se realiza el encriptado o des encriptado de los datos eficientemente. Luego de cifrar los datos, la codificación a Base 64 permitió una transferencia más limpia y confiable de los mismos a través de INTERNET. De esta forma y utilizando ambos algoritmos se lograron transferir datos manteniendo un nivel de seguridad lo suficientemente alto y acorde para la información manejada por la aplicación.

En cuanto a las emulaciones de la aplicación realizadas en los diferentes simuladores referentes a los modelos escogidos para realizar las pruebas (8900, 8520, 9630, 9550 y 9700), las mismas resultaron cumplir con los objetivos planteados. Ya que no se presentaron problemas de conectividad con el servidor, de igual forma el mismo estuvo en capacidad de recibir peticiones y dar la respuesta apropiada a estas. Los resultados arrojados producto de la ejecución de cada una de las etapas de la aplicación en el simulador, fueron iguales para cada dispositivo simulado.

Recomendaciones

Como recomendaciones se pueden destacar varios puntos. En principio, la aplicación desarrollada en este proyecto posee compatibilidad para dispositivos *BLACKBERRY* específicamente con SO 5.0, por lo que sería imprescindible para un futuro actualizar la aplicación para que pueda ser gestionada desde dispositivos *BLACKBERRY* con un sistema operativo superior al 5.0. Por otra parte, como ya se mencionó, entre los mecanismos de seguridad implementados para la aplicación está el algoritmo de cifrado Triple DES, que aunque ofrece un estándar de seguridad relativamente alto no es el más avanzado que existe en la actualidad, por lo que es recomendable utilizar, para futuras versiones de la aplicación, algoritmos más avanzados como lo es el AES (*Advanced Encryption Standard*). Finalmente se debe tomar en cuenta que la función de esta aplicación está enmarcada únicamente en ser

una herramienta para la supervisión de la información almacenada en servidores, por lo tanto, las tareas de administración y almacenamiento de dicha información en los servidores esta fuera del alcance de la aplicación, es por ello que se recomienda que los servidores cuenten con una gestión y administración local de los datos resguardados en los mismos.

LISTA DE REFERENCIAS

Common. (24 de Mayo de 2007). Solution Integration Guide for Multimedia Communication Server 5100/WLAN/Blackberry Enterprise Server. Canada: Nortel Networks.

Connors, A., & Sullivan, B. (14 de Diciembre de 2010). Mobile Web Application Best Practices. W3C Recommendation. Estados Unidos de America.

Esteban, A. (2000). *Tecnologías de Servidor con Java: Servlets, JavaBeans, JSP.* Madrid, España: Grupo EIDOS.

Fabris, A. (2009). *User Interface for Blackberry Smartphones.* Vancouver: Research In Motion.

Forouzan, B. A. (2007). *Transmisión de Datos y Resdes de Comunicaciones* (Cuarta Edición ed.). (I. S. Sánchez Gonzalez, Ed.) Madrid: McGraw-Hill/ Interamericana de España S. A. U.

Foust, B. (2010). *Blackberry Java Application Development.* Birmingham, Reino Unido: Packt Publishing Ltd.

García, C. A., & De Sousa, J. A. (11 de Enero de 2011). Redes WSN Para el Monitoreo de la Cadena de Frío. Caracas, Distrito Capital, Venezuela.

Gosling, J., Steele, G., & Bracha, G. (2005). *The Java Language Specification.* Addison-Wesley.

King, C. (2009). *Advance Blackberry 6 Development.* New York: Apress.

Kroll, M., & Haustein, S. (2002). *Java™ 2 Micro Edition Application Development.* Estados Unidos de America: Sams Publishing.

López Franco, J. M. (15 de Octubre de 2001). *Servidores de aplicaciones* . Recuperado el 8 de 5 de 2011, de http://trevinca.ei.uvigo.es/~txapi/espanol/proyecto/superior/memoria/node21.html

Riveros, F. (s.f.). *Introducción a la Arquitectura Blackberry.* Recuperado el 7 de Abril de 2011, de blackberry.devcite.com: http://blackberry.devcite.com/category/primeros-pasos-blackberry/

Rizk, A. (2009). *Beginning BlackBerry Development.* New York: Apress.

Rodriguez, J. I., García de Jalón, J., & Imaz, A. (1999). *Aprenda Servlets de Java Como si Estuviese en Primero.* San Sebastián: TECNUN.

Rojas, S., & Lucas, D. (2003). *Java a Tope:J2ME.* Málaga, España: Universidad de Málaga.

Serna, J. (10 de Enero de 2007). Redes de Sensores Inalámbricos. Barcelona, España.

Taylor, A., Buege, B., & Layman, R. (2002). *HACKERS de JAVA y J2ME, Desarrolle aplicaciones Java seguras.* (Primera Edición ed.). (C. Sánchez, Ed.) Madrid: McGraw-Hill.

Wiley, J. (4 de Julio de 2004). *Wireless Sensor Networks.* (D. C. Das, Ed.) Recuperado el 5 de Junio de 2011, de http://arri.uta.edu: http://arri.uta.edu/acs/networks/WirelessSensorNetChap04.pdf

ANEXOS

Anexo 1. Clase *Acceso*

```
package axis.etapas;

import java.io.IOException;

import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;


import net.rim.device.api.crypto.AESKey;
import net.rim.device.api.crypto.CryptoTokenException;
import net.rim.device.api.crypto.CryptoUnsupportedOperationException;
import net.rim.device.api.crypto.TripleDESKey;
import net.rim.device.api.system.Bitmap;
import net.rim.device.api.system.Characters;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.component.BitmapField;
import net.rim.device.api.ui.component.ButtonField;
import net.rim.device.api.ui.component.EditField;
import net.rim.device.api.ui.component.LabelField;
import net.rim.device.api.ui.component.PasswordEditField;
import net.rim.device.api.ui.component.SeparatorField;
import net.rim.device.api.ui.container.HorizontalFieldManager;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.container.VerticalFieldManager;

public class Acceso extends MainScreen implements FieldChangeListener
{}{
    BitmapField bitmapField;
    EditField campousuario;
    EditField respuesta;
    EditField campopin;
    PasswordEditField campocontra;
    ButtonField botonCont;

    public void fieldChanged(Field field, int context)
    {}{
        if (field == botonCont)
        {}{
            Logueo cd = new Logueo("Acceso","Ingrese Datos");
            UiApplication.getUiApplication().pushScreen(cd);
        }

    }

    public Acceso()
    {}{
        Bitmap logoBitmap = Bitmap.getBitmapResource("axis.PNG");
        bitmapField = new BitmapField(logoBitmap, Field.FIELD_HCENTER);

        add(bitmapField);

        add(new SeparatorField());

        botonCont = new ButtonField("Continuar",
ButtonField.CONSUME_CLICK|ButtonField.FIELD_HCENTER);
        VerticalFieldManager buttonManager = new
VerticalFieldManager(VerticalFieldManager.FIELD_HCENTER);
        buttonManager.add(botonCont);
        botonCont.setChangeListener(this);

        add(buttonManager);
```

Anexo 2. Clase *Actualizar*

```
package axis.etapas;

import net.rim.device.api.database.Database;
import net.rim.device.api.database.DatabaseFactory;
import net.rim.device.api.database.DatabaseIOException;
import net.rim.device.api.database.Row;
import net.rim.device.api.database.Statement;
import net.rim.device.api.io.URI;
import net.rim.device.api.system.Display;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.component.ButtonField;
import net.rim.device.api.ui.component.LabelField;
import net.rim.device.api.ui.component.RichTextField;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.container.VerticalFieldManager;

public class Actualizar extends MainScreen
{{
    String id = new String();

    public Actualizar()
    {{
        CreaBase base = new CreaBase();
        base.crea();
    }

    public void act(String[] vec, int nc)
    {{
        int i = 0;
        CreaTabla tablas = new CreaTabla();
        Datos dat = new Datos();
        while (i<nc)
        {{
            tablas.CreaCoor(vec[i]);
            tablas.CreaTemp(vec[i]);
            i++;
        }
        i=0;
        while (i<nc)
        {{
            ExtraeCoor exc = new ExtraeCoor(vec[i]);
            dat.insertacoor(exc.cadenades,vec[i]);
            ExtraeTemp ext = new ExtraeTemp(vec[i]);
            dat.insertatemp(ext.cadenades,vec[i]);
            i++;
        }
    }
}
```

Este método "*act*" recibe como parámetro de ejecución, "*vec*" y "*nc*". "*vec*" se refiere a un arreglo de tipo *String* que es obtenido a partir de la variable "*Ids*" de la clase "*CamCirc*". "*nc*" es el número de camiones en circulación obtenido a partir de

la variable "*cams*" de la misma clase. Se crea un nuevo objeto llamado "*tablas*" cuyo tipo pertenece a la clase "*CreaTabla*".

Anexo 3. Clase *CamCirc*

```
package axis.etapas;

import java.io.IOException;
import java.io.InputStream;
import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;

public class CamCirc {
public String[] Ids;
public String cadena;
public String cad;
int j = 0;
int cams;
    public CamCirc()
    {}{
        Crypto cr = new Crypto();
        String datosen = extrae();
        String datosdes = cr.decrypt(datosen);
        cams = num(datosdes);
        Ids = getID(datosdes,cams);
    }

    public int num(String cad){
        String cmp;
        int i = 0;
        char array[] = cad.toCharArray();
        while (i<=(array.length-1)){
            cmp = String.valueOf(array[i]);
            if (cmp.equals(",")){
                j++;
                i++;
            }
            i++;
        }
        return j;
    }
    public String[] getID( String cad, int camiones){
        int i = 0;
        int f = 0;
        String cmp="";
        String pal="";
        char array[] = cad.toCharArray();
        String[] camvec = new String[camiones];
        while (i<=(array.length-1))
        {}{
            cmp = String.valueOf(array[i]);
            if (cmp.equals(","))
            {}{
                camvec[f] = pal;
                f++;
                pal = "";
                cmp ="";
            }
            pal = pal+cmp;
            i++;

        }
        return camvec;
    }
    public String extrae()
    {}{
        String url = "http://axiserv.com:8080/Axisproy4/CamCirc";
        HttpConnection c = null;
        InputStream is = null;
```

```
    StringBuffer b = new StringBuffer();
    try
    {
        c = (HttpConnection)Connector.open(url);
        c.setRequestMethod(HttpConnection.GET);
        c.setRequestProperty("IF-Modified-Since", "20 Jan 2001 16:19:14 GMT");
        c.setRequestProperty("User-Agent",
        "Profile/MIDP-2.0 Configuration/CLDC-1.1");
        c.setRequestProperty("Content-Language", "en-CA");
        is = c.openDataInputStream();
        int ch;
        while ((ch = is.read()) != -1)
        {
              b.append((char) ch);
        }
        if(is!= null)
        {
            try
            {
                is.close();
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
         }
        if(c != null)
        {
            try
            {
                c.close();
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }
        cadena = b.toString();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
    return cadena;
}
```

Primero se crea una variable perteneciente a la clase "*Crypto*", que al igual que "*CamCirc*" es una clase diseñada en el proceso de desarrollo, la misma cumple funciones de encriptación y desencriptación de los datos que provee la aplicación web que se le provea a la misma (Clase "*Crypto*", véase Anexo 7).

Luego se define una variable de tipo *String* llamada "*datosen*", el valor de esta variable deriva de la ejecución de una función que retorna los datos referentes a la tabla "*camionescirc*" alojada en la base de datos. Dicha función tiene como nombre "*extrae*".

Como se puede observar en el código anterior, esta función establece una conexión de forma similar a la mostrada en el código de la función "*login*" de la clase "*Logueo*"). Con la diferencia en la URL que usa el conector para realizar la solicitud al servidor, ya que accesa a un *Servlet* llamado "*CamCirc*", el cual está incluido en la aplicación *web* que se ejecuta en el servidor. Luego se obtienen los datos encriptados provenientes del servidor y la función devuelve una variable de tipo *String* llamada "*cadena*" asignándole a la misma los datos recibidos.

En el constructor se define una variable llamada "*datosdes*" la cual contendrá los datos recibidos anteriormente ya decodificados por la función "*decrypt*" de la clase "*Crypto*". Estos datos son usados para obtener el número de camiones en circulación (asignados en la variable "*cams*" de tipo *int*) y la identificación perteneciente a cada uno de los camiones.

Para obtener el número de camiones se utiliza la rutina "*num*". Esta función, recibe como parámetro la cadena decodificada, la convierte a un arreglo de caracteres y examina el arreglo posición por posición en busca del separador de identificadores usado por el *Servlet* (","). Por cada "," (coma) encontrada suma uno a la variable "j" de tipo *int*. Luego de realizar el ciclo de búsqueda, devuelve el valor de la variable "j" que se refiere al número de camiones en circulación. Este valor se le asigna a la variable "*cams*".

Adicionalmente se define una variable llamada "*Ids*", la cual es un vector o arreglo de tipo *String*. El mismo contiene un identificador de camión por cada posición, es decir, el número de posiciones del vector es equivalente al número de

camiones en circulación. Los valores asignados a las posiciones propias de dicha variable dependen del algoritmo que define la función "*getID*" contenido en la clase "*CamCirc*".

Observando el código de la función "*getID*", se observa que la rutina tiene como parámetros de entrada "cad" (la cadena de caracteres decodificada resultado de la respuesta emitida por el servidor) y "*camiones*" (referente al número de camiones en circulación). Además se define un vector de tipo *String* llamado "*camvec*" que contendrá en cada posición un identificador de camión.

De forma similar a la rutina empleada para obtener el número de camiones ("*num*"), esta función convierte la cadena a un arreglo de caracteres para examinar cada caractér integrante de la cadena. Seguido de esta conversión, se ejecuta un ciclo el cual examina cada caractér. Si el caractér es diferente al separador (","), el mismo se concatena con el *String* "pal". Al encontrar una coma, se le asigna el valor de "*pal*" a la una posición del vector "*camvec*" en curso. El ciclo mencionado anteriormente es ejecutado hasta que se examine la última posición del arreglo referente a la cadena de caracteres ("*array*"). Al asignar cada identificador a cada posición del arreglo "*camvec*", la función retorna el valor del mismo y se le asigna a la variable "*Ids*".

Luego de que el usuario es autenticado, se crea esta variable "*cc*" y posteriormente se crea una variable de la clase Principal llamada "*pp*" y recibe como parámetros de entrada los parámetros descritos anteriormente ("*Ids*" y "*cams*"). La clase "*Principal*" es una de las pantallas que describen las etapas de la aplicación. Esta pantalla se encarga de mostrar los camiones de circulación y coloca un campo (botón) cuya etiqueta de identificación está relacionada con un identificador de camión. En cuanto al número de campos o botones visibles en la pantalla, dependerá del número de camiones que se encuentren en circulación según la información suministrada por la base de datos con la cual trabaja el sistema.

Anexo 4. Clase *CamScr*

```
package axis.etapas;

import java.util.Date;

import net.rim.device.api.database.DataTypeException;
import net.rim.device.api.database.Database;
import net.rim.device.api.database.DatabaseException;
import net.rim.device.api.database.DatabaseFactory;
import net.rim.device.api.database.DatabaseIOException;
import net.rim.device.api.database.DatabasePathException;
import net.rim.device.api.database.Row;
import net.rim.device.api.database.Statement;
import net.rim.device.api.i18n.DateFormat;
import net.rim.device.api.io.MalformedURIException;
import net.rim.device.api.io.URI;
import net.rim.device.api.system.Bitmap;
import net.rim.device.api.system.ControlledAccessException;
import net.rim.device.api.system.EncodedImage;
import net.rim.device.api.system.GIFEncodedImage;
import net.rim.device.api.ui.Color;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.Screen;
import net.rim.device.api.ui.TouchEvent;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.component.ButtonField;
import net.rim.device.api.ui.component.Dialog;
import net.rim.device.api.ui.component.LabelField;
import net.rim.device.api.ui.component.RichTextField;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.container.VerticalFieldManager;
import net.rim.device.api.ui.decor.BackgroundFactory;
import net.rim.device.api.ui.extension.component.PictureScrollField;
import net.rim.device.api.ui.extension.component.PictureScrollField.HighlightStyle;
import net.rim.device.api.ui.extension.component.PictureScrollField.ScrollEntry;

public class CamScr extends MainScreen  {
    Database d;
    String ide;
    String[] hr = new String[14];
    Muestra mues;
    PictureScrollField bancoIc;
    int z = 14;
    ButtonField act;
    public CamScr(String id)
    {{
        ide = id;
        hr = Posiciones.vec(id);
        ScrollEntry[] pos = new ScrollEntry[ 10 ];
        pos[0] = new ScrollEntry(Bitmap.getBitmapResource("mp.png"), "Posicion a las
"+hr[0],"");
        pos[1] = new ScrollEntry(Bitmap.getBitmapResource("mp.png"), "Posicion a las
"+hr[1],"");
        pos[2] = new ScrollEntry(Bitmap.getBitmapResource("mp.png"), "Posicion a las
"+hr[2],"");
        pos[3] = new ScrollEntry(Bitmap.getBitmapResource("mp.png"), "Posicion a las
"+hr[3],"");
        pos[4] = new ScrollEntry(Bitmap.getBitmapResource("mp.png"), "Posicion a las
"+hr[4],"");
        pos[5] = new ScrollEntry(Bitmap.getBitmapResource("mp.png"), "Posicion a las
"+hr[5],"");
        pos[6] = new ScrollEntry(Bitmap.getBitmapResource("mp.png"), "Posicion a las
"+hr[6],"");
        pos[7] = new ScrollEntry(Bitmap.getBitmapResource("mp.png"), "Posicion a las
```

```
"+hr[7],"");
        pos[8] = new ScrollEntry(Bitmap.getBitmapResource("mp.png"), "Posicion a las
"+hr[8],"");
        pos[9] = new ScrollEntry(Bitmap.getBitmapResource("mp.png"), "Posicion a las
"+hr[9],"");
        bancoIc = new PictureScrollField(150, 100);
        bancoIc.setData(pos, 0);
        bancoIc.setHighlightStyle(HighlightStyle.ILLUMINATE);
        bancoIc.setHighlightBorderColor(Color.RED);
        bancoIc.setHighlightStyle(HighlightStyle.ILLUMINATE_WITH_ROUND_BORDER);
        bancoIc.setBackground(BackgroundFactory.createSolidBackground(Color.GRAY));
        bancoIc.setLabelsVisible(true);
        add(bancoIc);
        bancoIc.setFocus();
        VerticalFieldManager vfm = new
VerticalFieldManager(VerticalFieldManager.FIELD_HCENTER);
        act = new ButtonField("Actualizar",ButtonField.CONSUME_CLICK);
        Date now = new Date();
        add(new LabelField("Ultima Actualizacion: "+
        DateFormat.getInstance(DateFormat.DATETIME_DEFAULT).format(now)));
        vfm.add(act);
        add(vfm);
    }

    protected boolean onSavePrompt()
    {
        return true;
    }

    protected boolean navigationClick(int status, int time)
    {
        if(bancoIc.isFocus())
        {
            show(bancoIc.getCurrentImageIndex());
            return true;
        }
         return super.navigationClick(status, time);
    }

    protected boolean touchEvent(TouchEvent message)
    {
        if(message.getEvent() == TouchEvent.CLICK)
        {
            if(bancoIc.isFocus())
            {
                show(bancoIc.getCurrentImageIndex());
                return true;
            }
        }

        return super.touchEvent(message);
    }
    public void show(int lab)
    {
        String compara = null;
        try
        {}{
            URI myURI = URI.create("file:///SDCard/Databases/Control.db");
            d = DatabaseFactory.open(myURI);
            Statement st = d.createStatement("SELECT * FROM camion"+ide+"coor");
            st.prepare();
            net.rim.device.api.database.Cursor c = st.getCursor();
            Row r;
            while (c.next())
```

```
            {}{
                r = c.getRow();
                try
                {}{
                    compara = (r.getString(3));
                }
                catch (DataTypeException e)
                {}{
                    e.printStackTrace();
                }
                if (compara.equals(hr[lab]))
                {}{
                    try
                    {}{
                        mues = new Muestra(r.getString(1),r.getString(2),z,ide);
                        UiApplication.getUiApplication().pushScreen(mues);
                    }
                    catch (DataTypeException e)
                    {}{
                        e.printStackTrace();
                    }
                }
            }
            st.close();
            d.close();
        }
        catch (DatabaseException e)
        {}{
            e.printStackTrace();
        }
        catch (IllegalArgumentException e)
        {}{
            e.printStackTrace();
        }
        catch (MalformedURIException e)
        {}{
            e.printStackTrace();
        }
    }
    public void Mapa()
    {
        UiApplication.getUiApplication().pushScreen(mues);
    }

}
```

El código referente al del constructor de la clase "*CamScr*" se define la creación del banco de iconos que representan los accesos a las posiciones representadas en un mapa geográfico. El mismo se crea a partir de la creación de un objeto perteneciente a la clase "*PictureScroollField*", al cual se llama "*bancoIc*".

"*PictureScrollField*" es una clase propia del JRE utilizado para el desarrollo de la aplicación.

Posteriormente se define un arreglo de tipo "*ScrollEntry*". El número de posiciones definidas para el arreglo, define el número de iconos presentes en "*bancoIn*". Ya que en pantalla se muestran las últimas 10 posiciones registradas, se define un arreglo de 10 posiciones.

Para obtener los datos referentes a las horas respectivas, se usa un arreglo de tipo *String* en el cual se guardan las horas registradas en la tabla ubicada en la base de datos "*Control*" referentes a las posiciones del camión para el cual se genero la pantalla "*CamScr*". Los valores correspondientes a las posiciones de dicho arreglo se obtienen con el llamado a la función "*vec*" perteneciente a la clase "*Posiciones*" (véase Anexo 18).

Si es seleccionado algún icono, se ejecuta el método "*show*" el cual recibe como parámetro, la posición del arreglo "*pos*" en la cual se encuentra el icono (el mismo está contenido en la variable "*lab*"). Ya que los números de posición de los elementos que conforman este arreglo coinciden con los elementos de la columna "*id*" de la tabla en donde se encuentran los datos de coordenadas ("*camionXcoor*"), se ejecuta en el método una búsqueda en la base de datos "*Control*" en la tabla referente al camión los parámetros latitud y longitud (columna 2 y 3 respectivamente) de la fila en la cual el parámetro id (en la tabla) sea igual al número de la posición del arreglo "*pos*" que recibe como parámetro "*show*".

Anexo 5. Clase *CreaBase*

```
package axis.etapas;

import net.rim.device.api.database.Database;
import net.rim.device.api.database.DatabaseFactory;
import net.rim.device.api.database.DatabaseIOException;
import net.rim.device.api.database.DatabasePathException;
import net.rim.device.api.io.MalformedURIException;
import net.rim.device.api.io.URI;
import net.rim.device.api.ui.component.LabelField;

public class CreaBase
{{
    Database d;
    URI myURI;
    public CreaBase()
    {{
    }
    public void crea()
    {{
        try
        {{
            myURI = URI.create("file:///SDCard/Databases/Control.db");
            d = DatabaseFactory.create(myURI);
            d.close();
        }
        catch (IllegalArgumentException e)
        {{
            e.printStackTrace();
        }
        catch (MalformedURIException e)
        {{
            e.printStackTrace();
        }
        catch (DatabaseIOException e)
        {{
            URI myURI;
            try
            {{
                myURI = URI.create("file:///SDCard/Databases/Control.db");
                DatabaseFactory.delete(myURI);
                CreaBase base = new CreaBase();
                base.crea();
            }
            catch (IllegalArgumentException e1)
            {{
                e1.printStackTrace();
            }
            catch (MalformedURIException e1)
            {{
                e1.printStackTrace();
            }
            catch (DatabaseIOException e1)
            {{
                e1.printStackTrace();
            } catch (DatabasePathException e1)
            {{
                e1.printStackTrace();
            }
        }
        catch (DatabasePathException e) {
            e.printStackTrace();
        }
    }
}
```

Detallando el código del método "*crea*", se crea un nuevo objeto de la clase URI llamado "*myURI*". El mismo se encarga de establecer la ruta personalizada para ubicar la base de datos en un directorio en específico ubicado en la memoria externa del dispositivo. Luego a la variable "*d*" se le asigna la base de datos a crear (la variable "*d*" es un objeto perteneciente a la clase *Database* contenida en el JRE de *BLACKBERRY*). Finalmente es creada la base de datos llamada "*Control*".

En caso de que en esta ruta se encuentre una base de datos con este nombre, la rutina emite una excepción del tipo *DatabaseIOException*. Si la condición anterior es cierta, se elimina la base de datos encontrada y se llama de nuevo al método "*crea*" para crear la nueva base de datos.

Anexo 6. Clase *CreaTabla*

```
package axis.etapas;

import net.rim.device.api.database.Database;
import net.rim.device.api.database.DatabaseException;
import net.rim.device.api.database.DatabaseFactory;
import net.rim.device.api.database.Statement;
import net.rim.device.api.io.MalformedURIException;
import net.rim.device.api.io.URI;
public class CreaTabla {
    Statement st;
    Database d;
    public  CreaTabla()
    {}{
    }
    public void CreaCoor(String id){
        try
        {}{
            URI myURI = URI.create("/SDCard/Databases/Control.db");
            d = DatabaseFactory.open(myURI);
            st = d.createStatement
            ( "CREATE TABLE 'camion"+id+"coor' ("+"'id' INTEGER, "
            + "'Lat' TEXT, " + "'Lon' TEXT, " + "'hr' TEXT )" );
            st.prepare();
            st.execute();
            st.close();
            d.close();
        }
        catch (DatabaseException e)
        {}{
            e.printStackTrace();
        }
        catch (IllegalArgumentException e)
        {}{
            e.printStackTrace();
        }
        catch (MalformedURIException e)
        {}{
            e.printStackTrace();
        }
    }
    public void CreaTemp(String id){
        try
        {}{
            URI myURI = URI.create("/SDCard/Databases/Control.db");
            d = DatabaseFactory.open(myURI);
            st = d.createStatement
            ( "CREATE TABLE 'camion"+id+"temp' ("+"'id' INTEGER, "
            + "'Temp' TEXT, " + "'hora' TEXT, " + "'tipo' TEXT )" );
            st.prepare();
            st.execute();
            st.close();
            d.close();
        }
        catch (DatabaseException e)
        {}{
            e.printStackTrace();
        }
        catch (IllegalArgumentException e)
        {}{
            e.printStackTrace();
        }
        catch (MalformedURIException e)
        {}{
            e.printStackTrace();
        }
    }
}
```

El método "*CreaCoor*" se encarga de crear una tabla referente a las coordenadas y el mismo recibe como parámetro el identificador del camión en curso mientras se ejecuta el ciclo. Este crea una tabla dentro de la base de datos "*Control*". El nombre de la tabla depende del identificador que recibe como parámetro este método (por ejemplo si la tabla se refiere al camión"**X**", la tabla tendrá como nombre "*camion**X**coor*"). Dicha tabla contiene 4 columnas:

- "*id*": esta columna contiene un entero entre 0 y 9, el mismo sirve de identificador para posición registrada.
- "*Lat*": referente a la latitud de la posición registrada.
- "*Lon*": referente a la longitud de la posición registrada.
- "*hr*": referente a la hora en que se registro esa posición en la base de datos.

El código para crear la tabla destinada a guardar los datos de temperatura, reside en el método "*CreaTemp*", y es similar al descrito anteriormente. Solo que la tabla cambiara el nombre (por ejemplo si la tabla se refiere al camión"**X**", la tabla tendrá como nombre "*camion**X**temp*").

Otra característica que la diferencia de la tabla anterior, es la función de las columnas que la conforman:

- "*id*": esta columna contiene un entero entre 0 y 9, el mismo sirve de identificador para la temperatura registrada.
- "*Temp*": temperatura registrada en grados centígrados.
- "*hora*": referente a la hora en que se registro esa posición en la base de datos.
- "*tipo*": define el tipo de cava (cava de refrigeración o cava de congelamiento).

Anexo 7. Clase *Crypto*

```
package axis.etapas;

import java.io.ByteArrayInputStream;

public class Crypto {
    private TripleDESKey _key;

    String keyStr = "2100000A2100000A2100000A";
    public Crypto() {
        _key = new TripleDESKey(keyStr.getBytes());

    }
     public String encrypt(String plaintext)
        {
            try
            {
                TripleDESEncryptorEngine encryptionEngine = new
 TripleDESEncryptorEngine(_key);

                PKCS5FormatterEngine formatterEngine = new PKCS5FormatterEngine(
                      encryptionEngine );

                ByteArrayOutputStream outputStream = new ByteArrayOutputStream();

                BlockEncryptor encryptor = new BlockEncryptor( formatterEngine,
                      outputStream );

                encryptor.write( plaintext.getBytes() );

                encryptor.close();

                byte[] encryptedData = outputStream.toByteArray();

                byte[] encoded = Base64OutputStream.encode(encryptedData, 0,
encryptedData.length, false, false);
                String strEncrypted = new String(encoded);
                return(strEncrypted);
            }
            catch( CryptoTokenException e )
            {
                System.out.println(e.toString());
            }
            catch (CryptoUnsupportedOperationException e)
            {
                System.out.println(e.toString());
            }
            catch( IOException e )
            {
                System.out.println(e.toString());
            }
            return "error";
        }

    public String decrypt(String ciphertext)
    {
            try
            {

            byte[] decoded = Base64InputStream.decode(ciphertext.getBytes(), 0,
ciphertext.length());

            TripleDESDecryptorEngine decryptorEngine =
                    new TripleDESDecryptorEngine(_key);
```

```
            PKCS5UnformatterEngine unformatterEngine =
                    new PKCS5UnformatterEngine( decryptorEngine );

            ByteArrayInputStream inputStream =
                    new ByteArrayInputStream( decoded );

            BlockDecryptor decryptor =
                    new BlockDecryptor( unformatterEngine, inputStream );

            byte[] temp = new byte[10];
            DataBuffer db = new DataBuffer();
            for( ;; )
            {
                int bytesRead = decryptor.read( temp );
                if( bytesRead <= 0 )
                {
                    break;
                }
                db.write(temp, 0, bytesRead);
            }

            byte[] decryptedData = db.toArray();
            String strDecrypted = new String(decryptedData);
            return(strDecrypted);
        }
        catch( CryptoTokenException e )
        {
            System.out.println(e.toString());
        }
        catch (CryptoUnsupportedOperationException e)
        {
            System.out.println(e.toString());
        }
        catch( IOException e )
        {
            System.out.println(e.toString());
        }
        return "error";
    }
}
```

Para el Encriptado/Desencriptado de los datos se decidió implementar el algoritmo por cifrado simétrico Triple DES, el cual a partir de una clave privada genera una llave secreta de 112 bits que utilizará el algoritmo para encriptar o des encriptar los datos. La primera parte del código muestra cómo se crea la llave secreta (utilizada por el algoritmo Triple DES) a partir de una clave privada.

Primero se inicializa la variable "*keyStr*", que corresponde a la clave privada conocida por el usuario, como un *String* de longitud 24. Luego, esta clave se transforma a un arreglo de 24 bytes para ser pasada como parámetro y generar la llave

secreta "*_key*" que utilizará el algoritmo durante el proceso de encriptación o des encriptación.

Ya con la llave creada sigue el proceso de encriptación de los datos a partir de la llave generada. La variable "*plaintext*" es el texto a encriptar que se pasa como parámetro. Primero se inicializa el motor de encriptación con la variable "*encrytionEngine*" que permite acceder directamente al algoritmo de Triple DES y que recibe como parámetro la llave secreta "*_key*" generada anteriormente. En la mayoría de los casos la data a encriptar no se ajusta al tamaño de los bloques de encriptación, ya que la longitud de dicha data debe ser múltiplo del tamaño de los bloques, se debe utilizar un algoritmo de relleno para asegurar que el flujo de datos que entra sea múltiplo del tamaño de los bloques de encriptación, en este caso se utilizó PKCS5 como algoritmo para rellenar los datos. Luego se llama al algoritmo de relleno con el nombre de "*formatterEngine*" y se le pasa como parámetro el motor de cifrado generado anteriormente, "*encrytionEngine*".

Luego se crea una variable de flujo de salida con el nombre de "*outputStream*", en la cual posteriormente se irá escribiendo la data encriptada. Ahora se debe generar un bloque de encriptación, "*encrytor*", pasándole como parámetros el "*formatterEngine* "que es el motor cifrado con algoritmo de relleno PKCS5, y la variable de flujo de salida "*outputStream*". La función de este bloque de encriptación es la de ir tomando la data que se va escribiendo en el flujo de salida y cifrarla según el algoritmo de encriptación que se especificó previamente, en este caso Triple DES.

Una vez que toda la cadena de bytes proveniente de la cadena de caracteres a encriptar "*plaintext*" pasa por el flujo de salida y es cifrada mediante los bloques de encriptado, se cierra dicho flujo de salida y se guarda toda la data codificada en una variable tipo byte llamada "*encryptedData*".

Antes de retornar el texto encriptado se le somete a una última codificación utilizando el algoritmo de Base64, esto con la finalidad de facilitar el envío de toda la cadena encriptada a través de internet.

Finalmente los datos ya cifrados, representados como un cadena de bytes encriptada y codificada a Base64, es transformada a un *String* como la variable "*strEncrypted*" la cual es retornada para luego ser enviada de forma segura a través de internet.

Para el proceso de des encriptado se debió desarrollar un código inverso al descrito anteriormente para encriptar. Primero, como la data que llega al dispositivo móvil está cifrada en Base64 se decodifica y se guarda en una variable del tipo byte llamada "*decoded*", en este punto dicha variable contiene un arreglo de bytes de los datos encriptados con Triple DES.

El siguiente paso es hacer un llamado a "*TripleDESDecryptorEngine*" para crear el motor de des encriptado que se nombró como "*decryptorEngine*", para hacer esto se debió pasar como parámetro la llave secreta "*_key*", que debe ser la misma que se utilizó en el proceso de encriptado.

Luego se debe generar una variable de tipo "*PKCS5UnformatterEngine*", que se nombró como "*unformatterEngine*", pasándole como parámetro el motor de des encriptado creado previamente, "*decryptorEngine*". Esto se hace con la finalidad de poder remover lo bytes de relleno que debieron ser agregados durante la encriptación de la data.

Lo siguiente es generar un flujo de entrada para el arreglo de bytes encriptados, para ello se crea la variable del tipo "*ByteArrayInputStream*" nombrada "*inputStream*", y se le pasa como parámetro el arreglo de bytes a des encriptar, "*decoded*".

Ahora se debe crear un bloque de des encriptado del tipo "*BlockDecryptor*" llamado "*decryptor*", pasándole como parámetro el motor de des encriptado "*unformatterEngine*" y el flujo de datos a des encriptar, "*inputStream*", de este modo toda la data que va siendo leída a través del bloque se desencripta automáticamente y puede ir siendo almacenada en un buffer mientras se va leyendo, se crea una variable

del tipo byte llamada "*temp*", con un tamaño específico de 10, y una variable del tipo buffer nombrada "*db*" en la cual se irá almacenando la data ya desencriptada a medida que se va leyendo desde el bloque de des encriptado.

El ciclo "*for*" dentro del procedimiento "*decryp*" cumple la función de ir leyendo la data contenida en el flujo de datos a través de bloque de des encriptado, dicha data la va leyendo de 10 en 10 bytes (esto se debe al tamaño específico que se le colocó a "*temp*") y a medida que la va leyendo esta se va desencriptando para luego ir siendo almacenada, también de 10 en 10 bytes, en la variable de tipo buffer "*db*". El ciclo termina de correr en el momento en que ya no hay más datos que leer, en este punto los datos des encriptados se encuentran almacenado en el buffer "*db*".

Por último toda la data decodificada y desencriptada almacenada en el buffer se pasa a un arreglo de bytes nombrado "*decryptedData*", para luego ser transformada en una variable del tipo *String*, "*strDecrypted*", la cual es finalmente retornada para poder trabajar con ella.

Anexo 8. Clase *Datos*

```
package axis.etapas;

import net.rim.device.api.database.Database;

public class Datos extends MainScreen  {
    Statement st;
    Database d;
    Database k;

    public Datos()
    {}{
    }
    public void insertacoor(String coor,String id)
    {}{
        int i = 0;
        String pal = "";String lat ="";
        String lon ="";
        String hr ="";
        String cmp="";
        int j=1;
        URI myURI;
        try
        {}{
            myURI = URI.create("file:///SDCard/Databases/Control.db");
            d = DatabaseFactory.open(myURI);
        }
        catch (IllegalArgumentException e1)
        {}{
            e1.printStackTrace();
        }
        catch (MalformedURIException e1)
        {}{
            e1.printStackTrace();
        }
        catch (ControlledAccessException e)
        {}{
            e.printStackTrace();
        }
        catch (DatabaseIOException e)
        {}{
            e.printStackTrace();
        }
        catch (DatabasePathException e)
        {}{
            e.printStackTrace();
        }
        char array[] = coor.toCharArray();
        while (i<=(array.length-1)&&(j<11))
        {}{
            cmp = String.valueOf(array[i]);
            if (cmp.equals(","))
            {}{
                if (lat.equals(""))
                {}{
                    lat = pal;
                    pal="";
                    cmp="";
                }
                else if (lon.equals(""))
                {}{
                    lon = pal;
                    pal="";
                    cmp="";
                }
```

```
                else if(hr.equals(""))
                {}{
                    hr = pal;
                    pal="";
                    cmp="";
                }
            }
            else if (cmp.equals(";"))
            {}{
                try
                {}{
                st = d.createStatement("INSERT INTO camion"+id+"coor(id,Lat,Lon,hr) "
                    + "VALUES ("+j+",'"+lat+"','"+lon+"','"+hr+"')");
                st.prepare();
                st.execute();
                j++;
                }
                catch ( Exception e )
                {}{
                    System.out.println( e.getMessage() );
                    e.printStackTrace();
                }
                lat="";
                lon="";
                pal="";
                hr="";
                cmp="";
            }
            pal = pal+cmp;
            i++;
        }
        try
        {}{
            st.close();
            d.close();
        }
        catch (DatabaseIOException e)
        {}{
            e.printStackTrace();
        }
        catch (DatabaseException e)
        {}{
            e.printStackTrace();
        }
    }
    public void insertatemp(String coor,String id)
    {}{ int i = 0;
        String pal = "";
        String temp ="";
        String hora ="";
        String tipo ="";
        String cmp="";
        int j=1;
        URI myURI;
        try
        {}{
            myURI = URI.create("file:///SDCard/Databases/Control.db");
            d = DatabaseFactory.open(myURI);
        }
        catch (IllegalArgumentException e1)
        {}{
            e1.printStackTrace();
        }
        catch (MalformedURIException e1)
```

```
{}{
    e1.printStackTrace();
}
catch (ControlledAccessException e)
{}{
    e.printStackTrace();
}
catch (DatabaseIOException e)
{}{
    e.printStackTrace();
}
catch (DatabasePathException e)
{}{
    e.printStackTrace();
}
char array[] = coor.toCharArray();
while (i<=(array.length-1)&&(j<11))
{}{
    cmp = String.valueOf(array[i]);
    if (cmp.equals(","))
    {}{
        if (temp.equals(""))
        {}{
            temp = pal;
            pal="";
            cmp="";
        }
        else if (hora.equals(""))
        {}{
            hora = pal;
            pal="";
            cmp="";
        }
        else if(tipo.equals(""))
        {}{
            tipo = pal;
            pal="";
            cmp="";
        }
    }
    else if (cmp.equals(";"))
    {}{
        try
        {}{
        st = d.createStatement("INSERT INTO camion"+id+"temp(id,Temp,hora,tipo)
                + "VALUES ("+j+",'"+temp+"','"+hora+"','"+tipo+"')");
        st.prepare();
        st.execute();
        j++;
        }
        catch ( Exception e )
        {}{
            System.out.println( e.getMessage() );
            e.printStackTrace();
        }
        temp="";
        hora="";
        pal="";
        tipo="";
        cmp="";
    }
    pal = pal+cmp;
    i++;
```

```
            }
            try {
                st.close();
                d.close();
            }
            catch (DatabaseIOException e) {
                e.printStackTrace();
            }
            catch (DatabaseException e) {
                e.printStackTrace();
            }
        }
    }
```

El procedimiento "*insertacoor*" posee como parámetros de entrada la cadena de caracteres que contienen la información acerca de las posiciones obtenidas del servidor y el identificador del camión. Se definen las variables respectivas para latitud, longitud y hora registrada de la posición. Luego se ejecuta el código respectivo para accesar a la base de datos "*Control*".

Luego se convierte la cadena a un arreglo de caracteres para poder realizar la comparación de cada carácter con los separadores y establecer a cual variable pertenecen los mismos. Los datos referentes a cada posición vienen representados de la siguiente manera: **latitud,longitud,hora;**.

El algoritmo recorre carácter por carácter y al encontrar una "," (coma) verifica en qué estado se encuentran las variables en el mismo orden en cómo se encuentran ordenadas en la cadena de datos. Si la variable está vacía, le asigna el valor que posee la variable "*pal*" a la misma; en caso contrario se concatena el carácter en "*pal*".

Al encontrar un carácter ";" (punto y coma), ya las variables "*lat*" (referente a la latitud), "*lon*" (referente a la longitud) y "*hr* "(hora de la posición registrada) están

con valores asignados, por tanto se ejecuta el código para llenar los valores respectivos a una fila de la tabla "*camionXcoor*" como se muestra a continuación.

Luego de realizar este proceso, las variables con las cuales se realiza el llenado de la tabla ("*lat", "lon","hr*") son devueltas a su valor inicial para realizar de nuevo las instrucciones explicadas anteriormente tantas veces como posiciones se encuentren en la cadena de datos suministrada al procedimiento.

Finalmente se ejecuta el código para cerrar la variable que realiza la sentencia para introducir los datos dentro de las filas correspondientes a la tabla de posiciones y para cerrar la conexión con la base de datos "Control".

Para la inserción de datos en la tabla de temperatura, el proceso de llenado es similar, dado que la rutina utilizada utiliza un código que se rige de la misma forma que el usado para el caso de las coordenadas. Solo se hacen cambios en el nombre de las variables a utilizar y el nombre de la tabla con la cual se ejecuta la sentencia para ingresar los datos en la misma.

La cadena (*String*) en la cual se encuentran los datos, es convertida a un arreglo de caracteres y se hace el mismo procedimiento para separa los parámetros para cada temperatura registrada. Cada registro de temperatura está definido de la siguiente manera: **temperatura,hora,tipo;.** Y las variables definidas para cada parámetro son las siguientes:

- "*temp*": se le asigna la temperatura registrada.
- "*hora*": se la asigna la hora en la cual se registro la temperatura correspondiente.
- "*tipo*": referente al tipo de cava.

Como se puede observar en el código anterior, al encontrar un";", se realiza la inserción de los datos en los campos definidos en la tabla correspondiente al camión. Luego de realizar la inserción de todos los datos se procede a cerrar la conexión con la base de datos.

Anexo 9. Clase *ExtraeCoor*

```
package axis.etapas;

import java.io.IOException;

public class ExtraeCoor
{}{
    static String cadena;
    String cadenades;
    String Id;
    public ExtraeCoor(String id)
    {}{
        Id =id;
        Crypto cr = new Crypto();
        String cad = extrae(Id);
        cadenades = cr.decrypt(cad);
    }

    public static String extrae(String id){
        String url = "http://axiserv.com:8080/Axisproy4/Coordenadas?id="+id;
        HttpConnection c = null;InputStream is = null;
        StringBuffer b = new StringBuffer();
        try  {
            c = (HttpConnection)Connector.open(url);
            c.setRequestMethod(HttpConnection.GET);
            c.setRequestProperty("IF-Modified-Since", "20 Jan 2001 16:19:14 GMT");
            c.setRequestProperty("User-Agent",
            "Profile/MIDP-2.0 Configuration/CLDC-1.1");
            c.setRequestProperty("Content-Language", "en-CA");
            is = c.openDataInputStream();  int ch;
            while ((ch = is.read()) != -1) {
                  b.append((char) ch);
            }
            if(is!= null) {
                try {
                    is.close();
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
             }
             if(c != null) {
                  try {
                    c.close();
                  }
                  catch (IOException e) {
                    e.printStackTrace();
                  }
              }cadena = b.toString(); }
        catch (IOException e)  {
            e.printStackTrace();
        }return cadena;}
}
```

Al recibir los datos encriptados producto de la solicitud hecha al servidor, los mismos son desencriptados por el método "*decrypt*", como se define en el constructor de la clase "*ExtraeCoor*".

Como se observa en el código anterior, "*cadenades*" es una variable de tipo *String* a la cual se le asigna los datos desencriptados. Los mismos serán usados posteriormente como parámetros de entrada en el método responsable por llenar los campos referentes a la base de datos ("*dat*" contenido en la clase "*Datos*") creada en la memoria externa del dispositivo móvil (véase Anexo 2).

Nótese como el parámetro "*id*" (identificador del camión) es usado en el parámetro *URL* que usa el objeto de tipo *HttpConnection* para establecer la conexión con el servidor y ejecutar la solicitud al *Servlet* .

Anexo 10. Clase *ExtraeTemp*

```
package axis.etapas;

import java.io.IOException;
import java.io.InputStream;

import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;

public class ExtraeTemp {
    static String cadena;
    String cadenades;
    String Id;
    public ExtraeTemp(String id) {
        // TODO Auto-generated constructor stub
        Id =id;
        Crypto cr = new Crypto();
        String cad = extrae(Id);
        cadenades = cr.decrypt(cad);
    }
    public static String extrae(String id)
    {}{
        String url = "http://axiserv.com:8080/Axisproy4/Temperatura?id="+id;
        HttpConnection c = null;
        InputStream is = null;
        StringBuffer b = new StringBuffer();
        try
        {
            c = (HttpConnection)Connector.open(url);
            c.setRequestMethod(HttpConnection.GET);
            c.setRequestProperty("IF-Modified-Since", "20 Jan 2001 16:19:14 GMT");
            c.setRequestProperty("User-Agent",
            "Profile/MIDP-2.0 Configuration/CLDC-1.1");
            c.setRequestProperty("Content-Language", "en-CA");
            is = c.openDataInputStream();
            int ch;
            while ((ch = is.read()) != -1)
            {
                  b.append((char) ch);
            }
            if(is!= null)
            {
                try
                {
                    is.close();
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
             }
             if(c != null)
             {
                  try
                  {
                    c.close();
                  }
                  catch (IOException e)
                  {
                    e.printStackTrace();
                  }
              }


               cadena = b.toString();
```

```
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
            return cadena;
        }
    }
```

Anexo 11. Clase *HrefField*

```
package axis.etapas;

import net.rim.device.api.ui.Color;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.Font;
import net.rim.device.api.ui.FontFamily;
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.component.Dialog;
import net.rim.device.api.system.Characters;


public class HrefField extends Field {

    private String content;
    private Font fieldFont;
    private int fieldWidth;
    private int fieldHeight;
    private boolean active = false;
    private int backgroundColour = 0xffffff;
    private int textColour = Color.BLACK;
    private int maskColour = Color.RED;

    private int buttonId;
    private String buttonName;

    public HrefField(String content)
    {
        super(Field.FOCUSABLE);
        this.content = content;
        fieldFont = defaultFont();
        fieldWidth = fieldFont.getAdvance(content)+16;
        fieldHeight = fieldFont.getHeight() + 10;

    }

    public void setColours(int backgroundColour, int textColour, int maskColour)
    {
        this.backgroundColour = backgroundColour;
        this.textColour = textColour;
        this.maskColour = maskColour;
        invalidate();
    }

    public void setBackgroundColour(int backgroundColour)
    {
        this.backgroundColour = backgroundColour;
        invalidate();
    }

    public void setTextColour(int textColour)
    {
        this.textColour = textColour;
        invalidate();
    }

    public void setMaskColour(int maskColour)
    {
        this.maskColour = maskColour;
        invalidate();
    }

    public void setFont(Font fieldFont)
    {
        this.fieldFont = fieldFont;
```

```
}

public int getPreferredWidth()
{
    return fieldWidth;
}

public int getPreferredHeight()
{
    return fieldHeight;
}

protected void layout(int arg0, int arg1)
{
    setExtent(getPreferredWidth(), getPreferredHeight());
}

protected void paint(Graphics graphics)
{


    if (active)
    {
        graphics.setColor(maskColour);
        graphics.setBackgroundColor(Color.GREEN);
        graphics.drawRoundRect(0, 0, fieldWidth, fieldHeight,1,1);
    }
    else
    {
        graphics.setColor(backgroundColour);
        graphics.drawRoundRect(0, 0, fieldWidth, fieldHeight,1,1);

    }

    graphics.setColor(textColour);
    graphics.setFont(fieldFont);
    graphics.drawText(content, 1, 1);

}

protected boolean navigationClick(int status, int time)
{
    fieldChangeNotify(1);
    Dialog.alert("hey");
    return true;
}

public void setButtonId(int buttonId)
{
    this.buttonId = buttonId;
}

public void setButtonName(String buttonName)
{
    this.buttonName = buttonName;
}

public int getButtonId()
{
    return buttonId;
}
public String getButtonName()
{
    return buttonName;
```

```
    }
    public boolean keyChar(char key, int status, int time)
    {
        if (key == Characters.ENTER)
        {
            fieldChangeNotify(0);
            return true;
        }

        return false;
    }


    protected void onFocus(int direction)
    {
        active = true;
        invalidate();
    }

    protected void onUnfocus()
    {
        active = false;
        invalidate();
    }

    public static Font defaultFont()
    {
        return Font.getDefault();
    }
}
```

Anexo 12. Clase *HrefFieldCamion*

```
package axis.etapas;

import net.rim.device.api.ui.Color;


public class HrefFieldCamion extends Field {

    private String content;
    private String id;
    private Font fieldFont;
    private int fieldWidth;
    private int fieldHeight;
    private boolean active = false;
    private int backgroundColour = Color.WHITE;
    //private int textColour = 0x333333;
    private int textColour = Color.WHITE;
    private int maskColour = Color.GREEN;
    private int c;
    private int buttonId;
    private String buttonName;

    public HrefFieldCamion(String content, String id)
    {
        super(Field.FOCUSABLE);
        this.content = content;
        this.id = id;
        try {
            FontFamily alphaSansFamily = FontFamily.forName("BBAlpha Serif");
            fieldFont = alphaSansFamily.getFont(Font.PLAIN, 14, Ui.UNITS_pt);
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        fieldWidth = 320;
        fieldHeight = 50;
    }

    public void setColours(int backgroundColour, int textColour, int maskColour)
    {
        this.backgroundColour = backgroundColour;
        this.textColour = textColour;
        this.maskColour = maskColour;
        invalidate();
    }

    public void setBackgroundColour(int backgroundColour)
    {
        this.backgroundColour = backgroundColour;
        invalidate();
    }

    public void setTextColour(int textColour)
    {
        this.textColour = textColour;
        invalidate();
    }

    public void setMaskColour(int maskColour)
    {
        this.maskColour = maskColour;
        invalidate();
    }
```

```
public void setFont(Font fieldFont)
{
    this.fieldFont = fieldFont;
}

public int getPreferredWidth()
{
    return fieldWidth;
}

public int getPreferredHeight()
{
    return fieldHeight;
}

protected void layout(int arg0, int arg1)
{
    setExtent(getPreferredWidth(), getPreferredHeight());
}

protected void paint(Graphics graphics)
{


    if (active)
    {
        graphics.setColor(maskColour);
        graphics.drawRoundRect(0, 0, fieldWidth, fieldHeight,1,1);
    }
    else
    {
        graphics.setColor(backgroundColour);
        graphics.drawRoundRect(0, 0, fieldWidth, fieldHeight,1,1);

    }

    graphics.setColor(textColour);
    graphics.setFont(fieldFont);
    graphics.drawText(content+id, 1, 5);



}

protected boolean navigationClick(int status, int time)
{
    fieldChangeNotify(1);
    Reportes rp = new Reportes(id);
    UiApplication.getUiApplication().pushScreen(rp);
    return true;
}

public void setButtonId(int buttonId)
{
    this.buttonId = buttonId;
}

public void setButtonName(String buttonName)
{
    this.buttonName = buttonName;
}

public int getButtonId()
{
```

```
        return buttonId;
    }
    public String getButtonName()
    {
        return buttonName;
    }
    public boolean keyChar(char key, int status, int time)
    {
        if (key == Characters.ENTER)
        {
            fieldChangeNotify(1);
            Dialog.alert("hey");
            return true;
        }

        return false;
    }


    protected void onFocus(int direction)
    {
        active = true;
        invalidate();
    }

    protected void onUnfocus()
    {
        active = false;
        invalidate();
    }

    public static Font defaultFont()
    {
        return Font.getDefault();
    }
}
```

La propiedad que permite seleccionar el campo definido por el código anterior, se le añade al campo o botón añadiendo la función "*navigationClick*" .

La misma permite que al seleccionar este campo, se cree un objeto del tipo "*Reportes*" al cual se llama como "*rp*" y posteriormente se ejecuta la sentencia para mostrar la pantalla en el dispositivo definida por esta clase "*UiApplication.getUiApplication().pushScreen(rp)".*

El parámetro "*id*" se refiere al identificador del camión que permite crear una etiqueta personalizada que depende del camión seleccionado para conocer las propiedades del mismo.

Anexo 13. Clase *HrefFieldPos*

```
package axis.etapas;

import net.rim.device.api.ui.Color;


public class HrefFieldPos extends Field {

    private String content;
    private String id;
    private Font fieldFont;
    private int fieldWidth;
    private int fieldHeight;
    private boolean active = false;
    private int backgroundColour = Color.WHITE;
    //private int textColour = 0x333333;
    private int textColour = Color.WHITE;
    private int maskColour = Color.GREEN;
    private int c;
    private int buttonId;
    private String buttonName;

    public HrefFieldPos(String content, String id)
    {
        super(Field.FOCUSABLE);
        this.content = content;
        this.id = id;
        try {
            FontFamily alphaSansFamily = FontFamily.forName("BBAlpha Serif");
            fieldFont = alphaSansFamily.getFont(Font.PLAIN, 14, Ui.UNITS_pt);
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        fieldWidth = 320;
        fieldHeight = 50;
    }

    public void setColours(int backgroundColour, int textColour, int maskColour)
    {
        this.backgroundColour = backgroundColour;
        this.textColour = textColour;
        this.maskColour = maskColour;
        invalidate();
    }

    public void setBackgroundColour(int backgroundColour)
    {
        this.backgroundColour = backgroundColour;
        invalidate();
    }

    public void setTextColour(int textColour)
    {
        this.textColour = textColour;
        invalidate();
    }

    public void setMaskColour(int maskColour)
    {
        this.maskColour = maskColour;
        invalidate();
    }
```

```
public void setFont(Font fieldFont)
{
    this.fieldFont = fieldFont;
}

public int getPreferredWidth()
{
    return fieldWidth;
}

public int getPreferredHeight()
{
    return fieldHeight;
}

protected void layout(int arg0, int arg1)
{
    setExtent(getPreferredWidth(), getPreferredHeight());
}

protected void paint(Graphics graphics)
{


    if (active)
    {
        graphics.setColor(maskColour);
        graphics.drawRoundRect(0, 0, fieldWidth, fieldHeight,1,1);
    }
    else
    {
        graphics.setColor(backgroundColour);
        graphics.drawRoundRect(0, 0, fieldWidth, fieldHeight,1,1);

    }

    graphics.setColor(textColour);
    graphics.setFont(fieldFont);
    graphics.drawText(content+id, 1, 5);



}

protected boolean navigationClick(int status, int time)
{
    fieldChangeNotify(1);
    CamScr cm = new CamScr(id);
    UiApplication.getUiApplication().pushScreen(cm);
    return true;
}

public void setButtonId(int buttonId)
{
    this.buttonId = buttonId;
}

public void setButtonName(String buttonName)
{
    this.buttonName = buttonName;
}

public int getButtonId()
{
```

```
        return buttonId;
    }
    public String getButtonName()
    {
        return buttonName;
    }
    public boolean keyChar(char key, int status, int time)
    {
        if (key == Characters.ENTER)
        {
            fieldChangeNotify(1);
            Dialog.alert("hey");
            return true;
        }

        return false;
    }


    protected void onFocus(int direction)
    {
        active = true;
        invalidate();
    }

    protected void onUnfocus()
    {
        active = false;
        invalidate();
    }

    public static Font defaultFont()
    {
        return Font.getDefault();
    }
}
```

Anexo 14. Clase *HrefFieldTemp*

```
kage axis.etapas;

ort net.rim.device.api.ui.Color;


lic class HrefFieldTemp extends Field {

 private String content;
 private String id;
 private Font fieldFont;
 private int fieldWidth;
 private int fieldHeight;
 private boolean active = false;
 private int backgroundColour = Color.WHITE;
 //private int textColour = 0x333333;
 private int textColour = Color.WHITE;
 private int maskColour = Color.GREEN;
 private int c;
 private int buttonId;
 private String buttonName;

 public HrefFieldTemp(String content, String id)
 {
     super(Field.FOCUSABLE);
     this.content = content;
     this.id = id;
     try {
         FontFamily alphaSansFamily = FontFamily.forName("BBAlpha Serif");
         fieldFont = alphaSansFamily.getFont(Font.PLAIN, 14, Ui.UNITS_pt);
     } catch (ClassNotFoundException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
     }

     fieldWidth = 320;
     fieldHeight = 50;
 }

 public void setColours(int backgroundColour, int textColour, int maskColour)
 {
     this.backgroundColour = backgroundColour;
     this.textColour = textColour;
     this.maskColour = maskColour;
     invalidate();
 }

 public void setBackgroundColour(int backgroundColour)
 {
     this.backgroundColour = backgroundColour;
     invalidate();
 }

 public void setTextColour(int textColour)
 {
     this.textColour = textColour;
     invalidate();
 }

 public void setMaskColour(int maskColour)
 {
     this.maskColour = maskColour;
     invalidate();
 }
```

```
public void setFont(Font fieldFont)
{
    this.fieldFont = fieldFont;
}

public int getPreferredWidth()
{
    return fieldWidth;
}

public int getPreferredHeight()
{
    return fieldHeight;
}

protected void layout(int arg0, int arg1)
{
    setExtent(getPreferredWidth(), getPreferredHeight());
}

protected void paint(Graphics graphics)
{


    if (active)
    {
        graphics.setColor(maskColour);
        graphics.drawRoundRect(0, 0, fieldWidth, fieldHeight,1,1);
    }
    else
    {
        graphics.setColor(backgroundColour);
        graphics.drawRoundRect(0, 0, fieldWidth, fieldHeight,1,1);

    }

    graphics.setColor(textColour);
    graphics.setFont(fieldFont);
    graphics.drawText(content+id, 1, 5);



}

protected boolean navigationClick(int status, int time)
{
    fieldChangeNotify(1);
    Temperatura tmp = new Temperatura(id);
    UiApplication.getUiApplication().pushScreen(tmp);
    return true;
}

public void setButtonId(int buttonId)
{
    this.buttonId = buttonId;
}

public void setButtonName(String buttonName)
{
    this.buttonName = buttonName;
}

public int getButtonId()
{
```

```
        return buttonId;
    }
    public String getButtonName()
    {
        return buttonName;
    }
    public boolean keyChar(char key, int status, int time)
    {
        if (key == Characters.ENTER)
        {
            fieldChangeNotify(1);
            Dialog.alert("hey");
            return true;
        }

        return false;
    }


    protected void onFocus(int direction)
    {
        active = true;
        invalidate();
    }

    protected void onUnfocus()
    {
        active = false;
        invalidate();
    }

    public static Font defaultFont()
    {
        return Font.getDefault();
    }
}
```

Anexo 15. Clase *Logueo*

```
package axis.etapas;

import java.io.IOException;

public class Logueo extends Screen implements FieldChangeListener {
    public String[] vec = new String[15];
    ButtonField cancelButton;
     ButtonField okButton;
     EditField nombre;
     PasswordEditField pass;
     TextField pin1;
     String Pin;
    GIFEncodedImage image;
    public Logueo(String titulo, String status) {
        super(new VerticalFieldManager());
        Pin =
(Integer.toString(net.rim.device.api.system.DeviceInfo.getDeviceId(),16)).toUpperCase
        add(new LabelField(""+titulo,LabelField.FIELD_HCENTER));
        add(new SeparatorField());
        add(new LabelField(""+status,LabelField.FIELD_HCENTER));
        add(new SeparatorField());
        nombre = new EditField("Nombre: ","");
        pass = new PasswordEditField("Contraseña: ","");
        add(nombre);
        add(pass);
        add(new LabelField("PIN: "+Pin));
        add(new SeparatorField());
        HorizontalFieldManager hfm = new
HorizontalFieldManager(HorizontalFieldManager.FIELD_HCENTER);
        okButton = new
ButtonField("Aceptar",ButtonField.CONSUME_CLICK|ButtonField.FIELD_HCENTER);
        okButton.setChangeListener(this);
        hfm.add(okButton);
        cancelButton = new
ButtonField("Cancelar",ButtonField.CONSUME_CLICK|ButtonField.FIELD_HCENTER);
        cancelButton.setChangeListener(this);
        hfm.add(cancelButton);
        add(hfm);
        add(new SeparatorField());

        }


    protected void sublayout(int width, int height) {
        // TODO Auto-generated method stub
        layoutDelegate(width - 80, height - 80);
        setPositionDelegate(10, 10);
        setExtent(width - 60, Math.min(height - 60, getDelegate().getHeight() + 20));
        setPosition(30, (height - getHeight())/2);
    }


    public void fieldChanged(Field field, int context) {
        // TODO Auto-generated method stub
        if (field == okButton) {


            login();
        }
        else if (field == cancelButton){
            System.exit(0);
        }
    }
```

```
    protected void paintBackground(Graphics graphics) {
        graphics.setColor(Color.BLACK);
        graphics.fillRoundRect(0, 0, getWidth(), getHeight(), 12, 12);
        graphics.setColor(Color.WHITE);
        graphics.drawRoundRect(0, 0, getWidth(), getHeight(), 12, 12);
        }
    public void  login() {
        String usuario = nombre.getText();
        String contra = pass.getText();
        String url = "http://axiserv.com:8080/Axisproy4/Acceso?nombre="+ usuario+
"&contra=" + contra + "&pin=" + Pin ;
        HttpConnection httpConn = null;
        try {
                httpConn = (HttpConnection) Connector.open(url+";deviceside=false");
        }
        catch (IOException e) {
                e.printStackTrace();
        }
        try {
            if (httpConn.getResponseCode() == HttpConnection.HTTP_OK){
                CamCirc cc = new CamCirc();
                Principal ppl = new Principal(cc.Ids,cc.cams);
                UiApplication.getUiApplication().pushScreen(ppl);
            }
            else
            {}{
                close();
                Logueo cd = new Logueo("Acceso Denegado","Intente Nuevamente");
                UiApplication.getUiApplication().pushScreen(cd);
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        try {
            httpConn.close();
        }
        catch (IOException e){
            e.printStackTrace();
        }
    }


}
```

En primer lugar se definen las variables de tipo *String* referentes a los parámetros escogidos para la validación del usuario dentro del sistema, los cuales son: "*nombre*", "*contra*" y "*pin*". El parámetro "nombre" se refiere al nombre del usuario, "contra" a la contraseña del mismo y "pin" al PIN referente al teléfono donde se ejecuta la aplicación. "nombre" y "contra" son introducidos por el usuario en la

pantalla de acceso, mientras que "pin" es obtenido a través de una API contenida en el JRE propio de BLACKBERRY, el cual viene integrado con el *Plug-in* mencionado en el Capítulo IV.

El código para asignar a la variable de tipo *String* "Pin" el valor del PIN (código único en cada teléfono inteligente BLACKBERRY) es el siguiente:

(*Integer.toString(net.rim.device.api.system.DeviceInfo.getDeviceId(),16)).toUpperCase()*

La API utilizada para la obtención del PIN es: *net.rim.device.api.system.DeviceInfo*

Luego de que se ejecuta la API, se convierte el valor obtenido (variable de tipo *Integer*) a una variable tipo *String*, posteriormente todos los caracteres se convierten en letras mayúsculas. Esta sentencia se ubica en el constructor de la clase.

Ya definidos los parámetros, se define una variable de tipo *String* llamada "*url*" la cual será el parámetro *URL* de la variable de tipo *HttpConnection* llamada "*httpCon*", necesario para establecer la conexión con el servidor y posteriormente realizar la ejecución del *Servlet* que realiza la tarea de autenticación del usuario.

Anexo 16. Clase *Minimizar*

```
package axis.etapas;

import java.util.Timer;
import java.util.TimerTask;

import net.rim.device.api.database.Database;
import net.rim.device.api.database.DatabaseFactory;
import net.rim.device.api.database.DatabaseIOException;
import net.rim.device.api.database.Row;
import net.rim.device.api.database.Statement;
import net.rim.device.api.io.URI;
import net.rim.device.api.system.Alert;
import net.rim.device.api.system.Application;
import net.rim.device.api.system.Bitmap;
import net.rim.device.api.ui.Ui;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.UiEngine;
import net.rim.device.api.ui.component.Dialog;

public class Minimizar
{{
    private Timer timer;
    int cont;
    int i;
    int comp;
    Database d;
    public Minimizar()
    {{
    }
    public void Alertas()
    {{
        close();
        timer = new Timer();
        timer.scheduleAtFixedRate(new TimerTask()
        {
            public void run()
            {

                tarea();
            }
        },10000, 10000);
    }
    protected void tarea()
    {
    UiApplication.getUiApplication().invokeLater(new Runnable()
    {
    public void run()
        {
        int z = 1;
        CamCirc cc = new CamCirc();
        Actualizar act = new Actualizar();
        act.act(cc.Ids,cc.cams);
        while (z<=cc.cams)
        {
        try
        {
            URI myURI = URI.create("file:///SDCard/Databases/Control.db");
            d = DatabaseFactory.open(myURI);
            Statement st = d.createStatement("SELECT id,Temp,hora,tipo " +
            "FROM camion"+z+"temp");
            st.prepare();
            net.rim.device.api.database.Cursor c = st.getCursor();
            Row r;
            while(c.next())
```

```
                {
                    r = c.getRow();
                    comp = Integer.parseInt(r.getString(1).trim());
                    if (r.getString(3).equals("ref"))
                    {
                        if (comp > 4 | comp < 1)
                        {
                            alerta(r.getInteger(0),r.getString(1),r.getString(2),z);
                        }
                    }
                    if (r.getString(3).equals("con"))
                    {
                        if (comp < -23 | comp > -18)
                        {
                            alerta(r.getInteger(0),r.getString(1),r.getString(2),z);
                        }
                    }
                }
                st.close();
                d.close();
                }
                catch ( Exception e ){
                e.printStackTrace();
                    }
                z++;
                    }
                }
            });
        }
        public void close() {
            UiApplication.getUiApplication().requestBackground();
        }
        public void alerta(final int i, final String tempe, final String hora,final int c){
        Application.getApplication().invokeLater(new Runnable() {
            public void run() {
                vibrate();
                UiEngine ui = Ui.getUiEngine();
                Dialog screen = new Dialog(Dialog.D_OK, "Camion: "+c+"Cava:"+i+
                " AlertaTemp:"+tempe+" Hora:"+hora,
                Dialog.OK, Bitmap.getPredefinedBitmap(Bitmap.EXCLAMATION),
                net.rim.device.api.ui.Manager.VERTICAL_SCROLL);
                ui.pushGlobalScreen(screen, 1, UiEngine.GLOBAL_QUEUE);
                timer.cancel();
                }
                private void vibrate() {
                    Alert.startVibrate(250);
                }
            });
        }

    }
```

Al seleccionarse la opción "*Minimizar*" dentro del menú ubicado en cada pantalla (definidas con las clases "Principal", "Temperatura", "*CamScr*", "*Muestra*", "*Reportes*"), se crea automáticamente un objeto de la clase "*Minimizar*". En la misma se ejecutan procesos donde se actualiza la base de datos "*Control*" periódicamente. Luego de la actualización se buscan valores dentro de las tablas de temperaturas de

los camiones operativos que se encuentren fuera del rango de temperaturas aceptables para emitir la alerta correspondiente.

Se ejecutan los métodos correspondientes en las clases "*CamCirc*" y "*Actualizar*" para obtener el número de camiones en circulación y actualizar las tablas que constituyen la base de datos "*Control*".

Posteriormente se ejecuta un ciclo para revisar las temperaturas contenidas en las tablas tantas veces como camiones en circulación existan.

Luego se comparan cada temperatura de la misma forma en cómo se comparan en la clase "*Temperatura*" (véase Anexo 21). Al encontrar una temperatura fuera del rango, la aplicación genera una notificación en la cual se informa al usuario el camión, la hora en que se registro el suceso, y el valor de la temperatura correspondiente mediante el método "alerta".

El informe de la alerta estará contenido en un dialogo descrito por la clase "*Dialog*" en la pantalla del dispositivo aun cuando el usuario no se encuentre realizando tareas dentro de la aplicación.

Para que el proceso de verificación de las tablas se ejecute periódicamente, se define un método llamado "*Alertas*". En el mismo se define el intervalo de tiempo para el cual se define la ejecución del método tarea. El mismo se define en la siguiente figura y está configurado para ejecutarse cada dos minutos (120000 micro segundos).

Anexo 17. Clase *Muestra*

```
package axis.etapas;

import java.io.IOException;
import java.io.InputStream;

import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;

import net.rim.device.api.system.Bitmap;
import net.rim.device.api.system.Display;
import net.rim.device.api.system.EncodedImage;
import net.rim.device.api.ui.Color;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.MenuItem;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.component.BitmapField;
import net.rim.device.api.ui.component.LabelField;
import net.rim.device.api.ui.component.Menu;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.container.VerticalFieldManager;
import net.rim.device.api.ui.decor.Background;
import net.rim.device.api.ui.decor.BackgroundFactory;
import net.rim.device.api.ui.extension.container.EyelidFieldManager;

public class Muestra extends MainScreen {
    HttpConnection connection = null;
    String result ="";
    BitmapField bitmapField;
    InputStream inputStream = null;
    VerticalFieldManager manager2;
    String latitud;
    String iden;
    String longitud;
    int zoom;
    public Muestra(String lat, String lon,int z,String ide){
        EyelidFieldManager manager = new EyelidFieldManager();
        manager.addTop(new CustomLabelField("Lat: "+lat+"° Long: "+lon+"°"));
        zoom = z;
        latitud =lat;
        longitud = lon;
        iden = ide;
        manager.setEyelidDisplayTime(10000);
        manager2 = (VerticalFieldManager)getMainManager();
        add(manager);

        try
         {
             connection = (HttpConnection) Connector.open
             ("http://maps.google.com/maps/api/staticmap?center="+latitud+","
            +longitud+"&zoom="+zoom+"&size="
            +Display.getWidth()+"x"+(Display.getHeight()-2)
            +"&maptype=roadmap&mobile=true&markers=color:blue|label:camion"+ide+"|"
            +lat+","+lon+"&sensor=false", Connector.READ, true);
             inputStream = connection.openInputStream();
             byte[] responseData = new byte[10000];
             int length = 0;
             StringBuffer rawResponse = new StringBuffer();
             while (-1 != (length = inputStream.read(responseData)))
             {
                 rawResponse.append(new String(responseData, 0, length));
             }
             int responseCode = connection.getResponseCode();
             if (responseCode != HttpConnection.HTTP_OK)
```

```
            {
                throw new IOException("HTTP response code: "  + responseCode);
            }

             result = rawResponse.toString();
    }catch (Exception e){}
    finally
    {
        try
        {
            inputStream.close();
            inputStream = null;
            connection.close();
            connection = null;
        }
        catch(Exception e){}
    }  try
    {
        byte[] dataArray = result.getBytes();
        EncodedImage bitmap;
        bitmap = EncodedImage.createEncodedImage(dataArray, 0, dataArray.length);
        final Bitmap googleImage = bitmap.getBitmap();
        bitmapField = new
BitmapField(googleImage,Field.STATUS_MOVE_FOCUS_HORIZONTALLY);
        Background bc = BackgroundFactory.createBitmapBackground(googleImage);
        manager.setBackground(bc);



    }
    catch (final Exception e){}

    }
      private static final class CustomLabelField extends LabelField
       {

           CustomLabelField(String text)
           {
               super(text);
           }

           public void paint(Graphics graphics)
           {
               graphics.setColor(Color.WHITE);
               super.paint(graphics);
           }
       }

      private MenuItem _closeMenu = new MenuItem("Cerrar", 110, 10) {
            public void run() {
                Minimizar minimizar = new Minimizar();
                minimizar.Alertas();
            }
        };

        private MenuItem _exitMenu = new MenuItem("Salir", 110, 10) {
            public void run() {
                exit();
            }
        };

        void exit() {
            System.exit(0);
        }
```

```
        private MenuItem _aleja = new MenuItem("Alejar", 110, 10) {
            public void run() {
                if (zoom>5){
                    zoom =(zoom - 1);
                    Muestra nm = new Muestra (latitud,longitud,zoom,iden);
                    UiApplication.getUiApplication().pushScreen(nm);

                }
            }
        };
        private MenuItem _acerca = new MenuItem("Acercar", 110, 10) {
            public void run() {
                if (zoom<16){
                    zoom =(zoom + 1);
                    Muestra nm = new Muestra (latitud,longitud,zoom,iden);
                    UiApplication.getUiApplication().pushScreen(nm);

                }
            }
        };
      protected void makeMenu( Menu menu, int instance) {
            menu.add(_closeMenu);
            menu.add(_exitMenu);
            menu.add(_aleja);
            menu.add(_acerca);
        }

}
```

El constructor de la clase recibe como parámetros *“lat”, “lon” y ”z”,* referentes a latitud de la posición, longitud de la misma y el nivel de acercamiento de la imagen respectivamente. La primera acción para descargar la imagen, es usar el API *Google Static Maps*. Dicho API es llamado al declarar el parámetro URL del objeto de tipo *HttpConnection* de siguiente modo.

"http://maps.google.com/maps/api/staticmap?center="+latitud+","+longitud +"&zoom="+zoom+"&size=”+Display.getWidth()+"x"+(Display.getHeight()2+"& maptype=roadmap&mobile=true&markers=color:blue|label:camion"+ide+"|" +lat+","+lon+"&sensor=false".

La ruta descrita anteriormente accesa ala rutina que genera el mapa estático (*http://maps.google.com/maps/api/staticmap)*, dicha rutina necesita ciertos parámetros para poder generar una imagen del área circundante al punto en donde se encuentra la coordenada, los cuales se separan de la ruta antes mencionada por “?” y

para diferenciar los valores de cada parámetro se utiliza el signo "&". Los parámetros son:

- *center*: se refiere a la coordenada geográfica representada en la imagen del mapa: Este parámetro se define en la URL presentada anteriormente como la latitud y longitud suministradas a la función.
- *zoom* : valor de acercamiento de la imagen (entre 0 y 23).
- size: tamaño en pixeles de la imagen a descargar. La imagen descargada es del tamaño de la pantalla del dispositivo, es por ello que se utilizan las funciones *Display.getWidth* (ancho de la pantalla del dispositivo en pixeles) y *Display.getHeight* (alto de la pantalla del dispositivo en pixeles).
- *maptype*: tipo de mapa usado para la imagen. En este caso se escoge "*roadtype*" ya que especifica una imagen de mapa de carreteras estándar.
- *mobile*: para determinar el tipo de dispositivo se está realizando la solicitud. Ya que se realiza la solicitud desde un dispositivo móvil, se coloca igual a "*true*".
- *markers*: establece el color del marcador de posición, la etiqueta del mismo y las coordenadas en dode se localizará el marcador dentro del mapa. En el caso de la url mostrada anteriormente, el marcador tiene color azul ("color:blue"), la etiqueta es el camión respectivo ("label:camión+ide+") y la ubicación del mismo, es la misma usada para el centro del mapa.

Luego el servidor envía la Imagen al dispositivo y es recibida por un objeto del tipo *InputStream* para luego ser alojada en un objeto StringBuffer ("stringBuffer") para luego ser convertido a un objeto de tipo *String* y asignar el valor del mismo a la variable "*result*".

Finalmente, se agrega la imagen a la pantalla con el siguiente código:

```
byte[] dataArray = result.getBytes();
EncodedImage bitmap;
bitmap = EncodedImage.createEncodedImage(dataArray, 0, dataArray.length);
final Bitmap googleImage = bitmap.getBitmap();
bitmapField = new BitmapField(googleImage,Field.STATUS_MOVE_FOCUS_HORIZONTALLY);
Background bc = BackgroundFactory.createBitmapBackground(googleImage);
manager.setBackground(bc);
```

Anexo 18. Clase *Posiciones*

```
package axis.etapas;

import net.rim.device.api.database.Database;
import net.rim.device.api.database.DatabaseFactory;
import net.rim.device.api.database.DatabaseIOException;
import net.rim.device.api.database.Row;
import net.rim.device.api.database.Statement;
import net.rim.device.api.io.URI;

public class Posiciones {
    static String vector[] = new String[14];
    static Database d;
    String ide;
    public Posiciones()
    {}{
    }
    public static String[] vec(String id){
        try
        {
            URI myURI = URI.create("file:///SDCard/Databases/Control.db");
            d = DatabaseFactory.open(myURI);
            Statement st = d.createStatement("SELECT id,Lat,Lon,hr FROM
camion"+id+"coor");
            st.prepare();
            net.rim.device.api.database.Cursor c = st.getCursor();
            Row r;
            int i = 0;
            while(c.next())
            {
                r = c.getRow();
                vector[i] = r.getString(3);
                i++;
            }

            st.close();
        }
        catch ( Exception e )
        {
            e.printStackTrace();
        }
        try {
            d.close();
        } catch (DatabaseIOException e) {
            e.printStackTrace();
        }
        return vector;
    }

}
```

La función "*vec*" ejecuta un ciclo en el cual extrae el elemento contenido en la columna "*hr*" de la tabla correspondiente a las posiciones del camión correspondiente al parámetro "*id*" suministrado a la función. El elemento extraído es de tipo *String* y contiene la hora registrada de la posición en el formato **hora:minutos**; al ser extraído, es asignado a una de las posiciones del arreglo "*vector*". Concluido el ciclo, "*vector*" contiene todas las horas contenidas en las filas que definen la tabla "*hr*" de "*camionXcoor*" (siendo X el identificador del camión). Como la función devuelve el objeto "*vector*" al concluir la rutina mencionada, los valores del mismo son asignados al arreglo definido en la clase "*CamScr*" llamado "*hr*".

Luego el vector es utilizado cuando se definen los parámetros de cada valor relativo a las posiciones del arreglo "*pos*". Por ejemplo, el icono definido en la posición 1 del arreglo (segundo icono) tendrá como etiqueta el dato contenido en la posición 1 de su similar "*hr*". Es por ello que en el campo en donde se inserta el parámetro que define el titulo se encuentra escrito de la siguiente manera: "Posicion a las +hr[1]".

Cada icono contenido en "*bancoIc*" posee la propiedad de dirigir la aplicación a realizar una actividad nueva, la misma consiste en la creación de un nuevo objeto de la clase "*Muestra*" la cual está diseñada para de generar una nueva pantalla en la aplicación. Además de generar una pantalla, hace usos de servicios ofrecidos por la empresa *Google*, ya que descarga una imagen estática de un mapa que mostrara geográficamente cualquier posición registrada en alguna de las tablas que almacenan información de este tipo en la base de datos "*Control*".

Anexo 19. Clase *Principal*.

```
package axis.etapas;

import axis.etapas.Minimizar;

public class Principal extends MainScreen   {


    ButtonField botonCont;
    private MenuItem cierra = new MenuItem("Minimizar", 110, 10) {
        public void run() {
            Minimizar minimizar = new Minimizar();
            minimizar.Alertas();
        }
    };

    private MenuItem salida = new MenuItem("Salir", 110, 10) {
        public void run() {
            exit();
        }
    };

    void exit() {
        System.exit(0);
    }

    protected void makeMenu( Menu menu, int instance) {
        menu.add(cierra);
        menu.add(salida);
    }

    public Principal(String[] vector,int numcams)
    {){
        int i = 0;

        setTitle("Principal - Camiones en Circulacion: "+numcams);
        CamCirc cc = new CamCirc();
        Actualizar actual = new Actualizar();
        actual.act(cc.Ids,cc.cams);

        while (i<=numcams-1){
            HrefFieldCamion fg = new HrefFieldCamion("Camion ",vector[i]);
            Background bc = BackgroundFactory.createSolidBackground(Color.GRAY);
            fg.setBackground(bc);
            add(fg);
            i++;
        }
    }

    public boolean keyChar(char key, int status, int time)
    {
        if (key == Characters.ESCAPE)
        {
            Acceso acceso = new Acceso();
            UiApplication.getUiApplication().pushScreen(acceso);
            return true;
        }

        return false;
    }


}
```

En el código del constructor de la clase "*Principal*", se puede observar el ciclo responsable de la colocación de los botones en esta pantalla. Este ciclo se repetirá tantas veces como posiciones posea el arreglo "vector". El botón que representa a cada camión es definido por la clase "*HrefFieldCamion*" (referenciado el código en el Anexo 12) y la misma obtiene como parámetro el identificador de cada camión para posteriormente reflejarla en la etiqueta del botón relacionado con cada camión.

Por otro lado, al observar el código anterior, es notorio la creación del objeto "*actual*" es cual lo define la clase "Actualizar". Esta clase contiene en su código el método o procedimiento "*act*", y este cumple la función de recolectar los datos correspondientes a las ultimas 10 temperaturas registradas en las tablas de las cavas de los camiones en circulación, del mismo modo se toman las ultimas 10 posiciones registradas en las tabla de posiciones que se encuentran en la base de datos referente a cada camión.

Anexo 20. Clase *Reportes*

```
package axis.etapas;

import net.rim.device.api.ui.Color;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.decor.Background;
import net.rim.device.api.ui.decor.BackgroundFactory;

public class Reportes extends MainScreen
{{

    public Reportes(String id)
    {{
        setTitle("Reportes - Camion "+id);
        HrefFieldTemp ht = new HrefFieldTemp("Temperaturas Camion",id);
        HrefFieldPos hp = new HrefFieldPos("Posiciones Camion",id);
        Background bc = BackgroundFactory.createSolidBackground(Color.GRAY);
        ht.setBackground(bc);
        hp.setBackground(bc);
        add(ht);
        add(hp);
    }
}
```

En el código del constructor de la clase anterior se definen dos objetos referentes a los campos:

- **"*ht*":** el campo lo define la clase "*HrefFieldTemp*", y es un campo seleccionable que al ser pulsado, redirige la aplicación a la pantalla donde se presentan las temperaturas asociadas al camión seleccionado en la pantalla definida por la clase "*Principal*".
- **"*hc*":** este campo está definido por la clase "*HrefFieldPos*". El mismo también posee la propiedad de ejecutar una acción al ser pulsado, y la acción consiste en redirigir la aplicación a la pantalla en la cual se presentan las posiciones añadidas a la tabla "*camionXcoor*" ubicada en la base de datos "*Control*".

Luego de ser creado los objetos pertenecientes las clases anteriormente mencionadas, son añadidas a la pantalla mediante las sentencias "*add(hc)"* y *"add(ht)"*.

Anexo 21. Clase *Temperatura*

```
package axis.etapas;

import java.util.Calendar;
import java.util.Date;

import net.rim.device.api.database.Database;
import net.rim.device.api.database.DatabaseFactory;
import net.rim.device.api.database.Row;
import net.rim.device.api.database.Statement;
import net.rim.device.api.i18n.DateFormat;
import net.rim.device.api.io.URI;
import net.rim.device.api.system.Characters;
import net.rim.device.api.ui.Color;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.XYEdges;
import net.rim.device.api.ui.component.ButtonField;
import net.rim.device.api.ui.component.Dialog;
import net.rim.device.api.ui.component.LabelField;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.container.VerticalFieldManager;
import net.rim.device.api.ui.decor.Background;
import net.rim.device.api.ui.decor.BackgroundFactory;
import net.rim.device.api.ui.decor.Border;
import net.rim.device.api.ui.decor.BorderFactory;
import net.rim.device.api.ui.picker.DateTimePicker;

public class Temperatura extends MainScreen implements FieldChangeListener {
    public String[] vec = new String[15];
    String ide;
    Database d;
    ButtonField act;
    VerticalFieldManager manager;
    int comp;
    DateFormat fr;
    public Temperatura(String id) {
        setTitle("Registro de Temperaturas - Camion "+id);
        ide=id;
        try
        {
            URI myURI = URI.create("file:///SDCard/Databases/Control.db");
            d = DatabaseFactory.open(myURI);
            Statement st = d.createStatement("SELECT id,Temp,hora,tipo FROM
camion"+id+"temp");
            st.prepare();
            net.rim.device.api.database.Cursor c = st.getCursor();
            Row r;
            int i = 0;
            while(c.next())
            {
                r = c.getRow();
                comp = Integer.parseInt(r.getString(1).trim());
                if (r.getString(3).equals("ref"))
                {
                    if (comp > 4 | comp < 1){
                        RepTemp fg = new RepTemp(""+r.getString(1)+"°C
@"+r.getString(2));
                        Background bc =
BackgroundFactory.createSolidBackground(Color.RED);
                        fg.setBackground(bc);
                        add(fg);
                    }
                    else{}{
```

```
                        RepTemp fg = new RepTemp(""+r.getString(1)+"°C
@"+r.getString(2));
                        Background bc =
BackgroundFactory.createSolidBackground(Color.GREEN);
                        fg.setBackground(bc);
                        add(fg);
                    }
                }
                else if (r.getString(3).equals("con")){
                    if (comp < -23 | comp > -18){
                        RepTemp fg = new RepTemp(""+r.getString(1)+"°C
@"+r.getString(2));
                        Background bc =
BackgroundFactory.createSolidBackground(Color.RED);
                        fg.setBackground(bc);
                        add(fg);
                    }
                    else {
                        RepTemp fg = new RepTemp(""+r.getString(1)+"°C
@"+r.getString(2));
                        Background bc =
BackgroundFactory.createSolidBackground(Color.GREEN);
                        fg.setBackground(bc);
                        add(fg);
                    }
                }


            }

            st.close();
            d.close();
        }
        catch ( Exception e )
        {

            e.printStackTrace();
        }
        VerticalFieldManager vfm = new
VerticalFieldManager(VerticalFieldManager.FIELD_HCENTER);
        act = new ButtonField("Actualizar",ButtonField.CONSUME_CLICK);

        Date now = new Date();
        add(new LabelField("Ultima Actualizacion: "+
        DateFormat.getInstance(DateFormat.DATETIME_DEFAULT).format(now)));



        vfm.add(act);
        add(vfm);

        act.setChangeListener(this);

    }

    public void fieldChanged(Field field, int context) {
        if (field == act){
            CamCirc cc = new CamCirc();

            Actualizar actual = new Actualizar();
            actual.act(cc.Ids,cc.cams);
            Temperatura te = new Temperatura(ide);
            UiApplication.getUiApplication().pushScreen(te);
```

Como paso principal, se establece la conexión con la base de datos "*Control*" localizada en la memoria externa del dispositivo. Como se puede ver en el código, la clase recibe como parámetro de trabajo "*id*" (identificador del camión). De este parámetro depende la tabla de la cual se obtendrán los registros de temperatura para ubicarlos en la pantalla.

Luego al objeto "*st*" se le asignan los datos pertenecientes a la base accesada. "*c*" es un objeto con el cual se accesa a cada una de las filas pertenecientes a dicha tablas. Y se ejecuta el ciclo para mostrar en pantalla las temperaturas.

El ciclo se ejecuta mientras el cursor "*c*" encuentre líneas nuevas en la tabla. Se define una variable de tipo *Integer* (entero) a la cual se le asigna la temperatura de la fila en la cual se encuentre el cursor. Este paso se hace posible gracias a la función *pareseInt* definida en la clase *Integer*. Posteriormente se verifica el tipo de cava a la cual pertenece la temperatura, el identificador para las cavas de refrigeración es "*ref*" y para las cavas de congelamiento "*con*".

Al encontrarse en una fila, primero se verifica el tipo de cava al cual está asociada la temperatura, si la cava es del tipo "*ref*", verifica que la temperatura se encuentre en 1 y 4; de ser cierto lo anterior, se crea un nuevo objeto perteneciente a la clase "RepTemp" , el cual gráficamente es una etiqueta rectangular en el cual se presentan los datos relacionados con la temperatura y la hora en la cual se registró la misma. Este objeto tendrá un fondo color verde.

Si por el contrario, la temperatura esta fuera del rango mencionado anteriormente, de igual forma se crea un objeto de la clase "RepTemp", pero el mismo tendrá un fondo color rojo.

Para definir el fondo de la etiqueta, se define un objeto de la clase "*Background*" ("bc"). Luego se hace uso de la función "createSolidBackground" contenida en la clase "BackgroundFactory". Dicha función posee un parámetro de entrada, el cual representa el color deseado para el fondo: para definir el color rojo el

parámetro se debe declarar el parámetro como “Color.RED”, si el fondo es de color verde, entonces se escribe “Color.GREEN”.

Luego se le añade el color a la etiqueta (“fg.setBackground(bc)”) y se añade la misma a la pantalla con la sentencia “add (fg)”.

Adicionalmente es declarado un botón de actualización, el botón es creado a partir de un objeto llamado “*act*” del tipo *ButtonField*, al ser seleccionado se ejecutan los pasos descritos en el apartado Actualización de Variables de la Aplicación ubicado en este capítulo, para crear una nueva base de datos, las tablas respectivas y actualizar las mismas con nueva información. Posteriormente se crea un objeto del tipo “*Temperatura*” y se muestra la nueva pantalla.

```
        }
    }
    public boolean keyChar(char key, int status, int time)
    {
        if (key == Characters.ESCAPE)
        {
            CamCirc cc = new CamCirc();

            Principal ppl = new Principal(cc.Ids,cc.cams);
            UiApplication.getUiApplication().pushScreen(ppl);
            return true;
        }

        return false;
    }

}
```

Anexo 22. Clase *Acceso* (Aplicación web).

```
package axisproy.clases;

import java.io.IOException;


/**
 * Servlet implementation class Acceso
 */
public class Acceso extends HttpServlet {
    private static final long serialVersionUID = 1L;
    static Connection conexion = null;
    static ResultSet rs = null;
    static Statement s = null;
    Boolean acept = false;
    Encripta enc;
    public static void conecta()
    {}{
        try
        {}{
           Class.forName("com.mysql.jdbc.Driver");
        }
        catch (Exception e)
        {}{
           e.printStackTrace();
        }
        try
        {}{
            conexion = DriverManager.getConnection
 ("jdbc:mysql://localhost/axis","root", "holahola");
        }
        catch (SQLException e)
        {}{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    public static void consulta()
    {}{
        try
        {}{
            s = conexion.createStatement();
        }
        catch (SQLException e)
        {}{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        try
        {}{
             rs = s.executeQuery ("select * from regis");
        }
        catch (SQLException e)
        {}{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public static void desconecta ()
    {}{
        try
        {}{
            conexion.close();
        }
```

```
        catch (SQLException e)
        {}{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    /**
     * @see HttpServlet#HttpServlet()
     */


    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        // TODO Auto-generated method stub
        response.setContentType("text/plain");

        String nombre=request.getParameter("nombre");
        String contra=request.getParameter("contra");

        String pin=request.getParameter("pin");
        conecta();
        consulta();
        try {
            while (rs.next()){
                if (rs.getString(2).equalsIgnoreCase(nombre) &&
rs.getString(3).equalsIgnoreCase(contra) && rs.getString(4).equalsIgnoreCase(pin)){
                    acept = true;
                }
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        if (acept){
            response.sendError(HttpServletResponse.SC_OK);
        }
        else{}{
            response.sendError(HttpServletResponse.SC_NOT_ACCEPTABLE);
        }
        acept = false;
        desconecta();

    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request,response);
    }


}
```

En el método “doGet”, se extraen los parámetros “nombre”, “contra” y “pin”, para luego asignar dichos valores a variables de tipo *String* definidas con el mismo nombre de los parámetros anteriores.

Luego se ejecuta el procedimiento “*conecta*”, el cual se encarga de establecer la conexión entre el servidor y la base de datos.

Posteriormente se ejecuta la búsqueda en la tabla que contiene la información referente a los usuarios que poseen el privilegio de acceso a los datos suministrados por la aplicación móvil (la tabla pertenece a la base de datos con la que se establece la conexión). Dicha tabla posee el nombre “*regis*”, la misma contiene 3 columnas referentes a los parámetros antes mencionados. El código para realizar la extracción de los valores de esta tabla es llamado “*consulta*”.

Este procedimiento le asigna a la variable “*rs*” de tipo *ResultSet,* la tabla completa que contiene la información de los usuario para posteriormente ejecutar la rutina de comparación que se encuentra seguida de ejecutar el procedimiento “*consulta*”. Esta rutina que se ejecuta mientras se encuentren filas de filas en la tabla sin examinar. Se comparan los valores de cada fila de la tabla con los valores obtenidos luego de la petición ejecutada por el dispositivo. Si los 3 parámetros coinciden con los parámetros que se encuentran en alguna de las filas, el servidor emite un código de respuesta *OK* (200), si por el contrario no se encuentran coincidencias con los parámetros obtenidos, el servidor emite una respuesta *NOT ACCEPTABLE* (406)

Anexo 23. Clase *Coordenadas* (Aplicación web).

```
package axisproy.clases;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Coordenadas
 */
public class Coordenadas extends HttpServlet {
    private static final long serialVersionUID = 1L;
    static Connection conexion = null;
    static ResultSet rs = null;
    static Statement s = null;
    public static void conecta()
    {{
        try
        {{
           Class.forName("com.mysql.jdbc.Driver");
        }
        catch (Exception e)
        {{
           e.printStackTrace();
        }
        try
        {{
            conexion = DriverManager.getConnection
("jdbc:mysql://localhost/axis","root", "holahola");
        }
        catch (SQLException e)
        {{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    public static void consulta(String id)
    {{
        try
        {{
            s = conexion.createStatement();
        }
        catch (SQLException e)
        {{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        try
        {{
             rs = s.executeQuery ("select * from camion"+id+"coor");
        }
        catch (SQLException e)
        {{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

```
    }

    public static void desconecta ()
    {}{
        try
        {}{
            conexion.close();
        }
        catch (SQLException e)
        {}{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    /**
     * @see HttpServlet#HttpServlet()
     */


    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        // TODO Auto-generated method stub
        response.setContentType("text/plain");
        String id = request.getParameter("id");
        PrintWriter out = response.getWriter();
        conecta();
        consulta(id);
        String cad="";
        try {
            while (rs.next()){
                cad = ((cad)+(rs.getString(2)+"," +rs.getString(3)+","
                        +rs.getString(4)+",;"));
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        Encripta en = new Encripta();
        String caden = en.encrypt(cad);
        out.write(caden);

        desconecta();
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request,response);
    }
}
```

Observando el código anterior, es notorio que el método "consulta" se encuentra ligeramente modificado para establecer dinamismo al momento de realizar la consulta y extracción de los datos contenidos en las tablas. Se le añade un parámetro llamado "id" que hace referencia al identificador del camión para poder extraer los datos de un camión en especifico. El dinamismo viene dado por la capacidad de reutilizar el mismo *Servlet* independientemente del camión por el cual se realiza la petición, ya que el procedimiento hace uso del identificador ("id") del camión para la extracción de los datos de la tabla requerida.

Otro aspecto a destacar es la forma en cómo se envían los datos al dispositivo. Cada elemento de la fila respectiva a la tabla es separado con una "," para diferenciarlo. Y luego de concatenar estos elementos (latitud, longitud y hora en que se registro la posición respectiva), se separa con ";" para indicar el fin de la fila. De esta forma se ejecuta el ciclo hasta recolectar todos los datos requeridos. Luego los parámetros concatenados son encriptados por el método "*encrypt*" descrito en el objeto "en" (el cual pertenece a la clase "*Encripta*"). Dicha clase se explica al final del capítulo.

Anexo 24. Clase *CamCirc* (Aplicación web).

```
package axisproy.clases;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Coordenadas
 */
public class CamCirc extends HttpServlet {
    private static final long serialVersionUID = 1L;
    static Connection conexion = null;
    static ResultSet rs = null;
    static Statement s = null;
    public static void conecta()
    {}{
        try
        {}{
           Class.forName("com.mysql.jdbc.Driver");
        }
        catch (Exception e)
        {}{
           e.printStackTrace();
        }
        try
        {}{
            conexion = DriverManager.getConnection
("jdbc:mysql://localhost/axis","root", "holahola");
        }
        catch (SQLException e)
        {}{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    public static void consulta()
    {}{
        try
        {}{
            s = conexion.createStatement();
        }
        catch (SQLException e)
        {}{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        try
        {}{
             rs = s.executeQuery ("select * from camionescirc");
        }
        catch (SQLException e)
        {}{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

```
    }

    public static void desconecta ()
    {}{
        try
        {}{
            conexion.close();
        }
        catch (SQLException e)
        {}{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    /**
     * @see HttpServlet#HttpServlet()
     */


    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        conecta();
        consulta();
        String cad="";
        try {
            while (rs.next()){
                cad = (cad+ rs.getString(1)+",");
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        Encripta en = new Encripta();
        String caden = en.encrypt(cad);
        out.write(caden);
        desconecta();
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request,response);
    }
}
```

Como se observa en el código anterior, se define una variable del tipo *PrintWriter* que lleva como atributo la respuesta del *Servlet*. Posteriormente se ejecutan los procedimientos de forma similar al *Servlet* explicado en el Anexo 22,

variando tabla a la que se accesa en el procedimiento consulta ya que la sentencia para obtener el acceso a la tabla que contiene los datos referentes a la tabla de camiones en circulación, seria:

rs = *s*.executeQuery ("select * from camionescirc");

Luego se copian en la variable "*out*" los identificadores de cada camión separados por "," para diferenciarlos entre ellos.

Anexo 25. Clase *Encripta* (Aplicación web).

```
package axisproy.clases;

import java.security.NoSuchAlgorithmException;

import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;

public class Encripta {
      public static final String DESEDE_ENCRYPTION_SCHEME = "DESede";
        private Cipher cipher;
        byte[] keyAsBytes;
        SecretKeySpec key;
    public Encripta() {
        // TODO Auto-generated constructor stub
        byte [] seed_key = (new String("2100000A2100000A2100000A")).getBytes();
        key = new SecretKeySpec(seed_key,"TripleDES");
        try {
            cipher=Cipher.getInstance("TripleDES");
        } catch (NoSuchAlgorithmException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (NoSuchPaddingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    public String encrypt(String unencryptedString) {
        String encryptedString = null;
        try {
            cipher.init(Cipher.ENCRYPT_MODE, key);
            byte[] plainText = unencryptedString.getBytes();
            byte[] encryptedText = cipher.doFinal(plainText);
            BASE64Encoder base64encoder = new BASE64Encoder();
            encryptedString = base64encoder.encode(encryptedText);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return encryptedString;
    }
     public String decrypt(String encryptedString) {
            String decryptedText=null;
            try {
                cipher.init(Cipher.DECRYPT_MODE, key);
                BASE64Decoder base64decoder = new BASE64Decoder();
                byte[] encryptedText = base64decoder.decodeBuffer(encryptedString);
                byte[] plainText = cipher.doFinal(encryptedText);
                decryptedText= bytes2String(plainText);
            } catch (Exception e) {
                e.printStackTrace();
            }
            return decryptedText;
        }
     private static String bytes2String(byte[] bytes) {
            StringBuffer stringBuffer = new StringBuffer();
            for (int i = 0; i < bytes.length; i++) {
                stringBuffer.append((char) bytes[i]);
            }
            return stringBuffer.toString();
        }
```

Esta clase corre del lado del servidor para poder encriptar o des encriptar los datos que salen o llegan al mismo, en general esta clase cumple las mismas funciones de que la clase "*Crypto"* (véase Anexo 7), la diferencia radica en que el código JAVA utilizado es diferente, esto se debe a que los códigos para desarrollar un servlet corriendo en un servidor web no son iguales que los utilizados en el desarrollo para aplicaciones móviles.

Como se observa en el código siguiente lo primero que se hace es crear una llave secreta a partir de una clave privada conocida por el usuario, es importante destacar que las claves privadas utilizadas tanto del lado del servidor como del dispositivo móvil deben ser las mismas, en este caso se utiliza como clave "2100000A2100000A2100000A".

La llave secreta se genera especificando el algoritmo a utilizar que en nuestro caso es "*TripleDES*" y se le pasa como parámetro un arreglo de byte, "*seed_key*", que contiene la clave privada. La llave secreta se guarda en una variable de tipo "*SecretKeySpec*" nombrada como "*key*", la cual se utiliza posteriormente para poder encriptar o desencriptar los datos.

La variable "*cipher*" del tipo "*Cipher*" es creada y proporciona funcionalidades de encriptación y des encriptación, utilizando el método "*.getInstance("TripleDES")*" se especifica que el algoritmo a utilizar para procesos de encriptación/desencriptación realizados por "cipher" será el TripleDES.

El código desarrollado para encriptar los datos se observa dentro del procedimiento "*encrypt*". En primer lugar, se recibe como parámetro la cadena "*unencryptedString*" que contiene los datos a encriptar. Luego se crea una variable llamada "*encryptedString*" en la que posteriormente se guardarán los datos encriptados.

Lo siguiente es inicializar el "*cipher*" mediante un llamado a ".init(Cipher.ENCRYPT_MODE, key)", de esta manera se inicia el "*cipher*" para un

modo de operación de encriptado y se le pasa como parámetro la llave secreta "*key*" previamente generada.

Luego se trasforma el string que contiene los datos a encriptar, "*unencryptedString*", en una variable de tipo byte nombrada como "*plainText*". El próximo paso es encriptar los datos contenidos en "plainText" mediante un llamado a "*cipher.doFinal*(*plainText*)", esto genera un arreglo de bytes ya encriptados que se guardan en otra variable del tipo byte llamada "*encryptedText*".

Finalmente se debe tomar el arreglo de bytes ya encriptados y codificarlos a Base 64. El resultado de esta codificación se almacena en la variable "*encryptedString*" la cual es retornada para luego poder enviar los datos ya encriptados y codificados.

El procedimiento "*decrypt*" muestra el proceso para des encriptar los datos que llegan desde el dispositivo móvil a el servidor. Primero se recibe como parámetro el string "*encryptedString*" que contiene los datos encriptados, luego se inicializa una variable en la que se almacenará la data ya des encriptada.

Al igual que en el proceso de encriptado se utiliza la variable cipher, con la diferencia que esta vez se inicializa mediante el código: "cipher.init(Cipher.DECRYPT_MODE, key)", se pasa como parámetro la llave secreta "key" que es la misma utilizada durante el proceso de encriptado, y se especifica que se utilizará el modo de des encriptado.

Como los datos contenidos en "encryptedString" están encriptados y codificados a Base64, primero deben ser decodificados mediante el código "base64decoder.decodeBuffer(encryptedString)", los datos decodificados se almacenan en una variable del tipo byte llamada "encryptedText" para luego ser des encriptados mediante "cipher.doFinal(encryptedText)", de este modo se genera un arreglo de bytes ya des encriptados que son almacenados en la variable "*plainText*".

Por último se debe tomar el arreglo de bytes con los datos ya des encriptados y transformarlos a una variable de tipo *String*, para esto se hace un llamado al proceso "*bytes2String*" que se observa a continuación:

```
private static String bytes2String(byte[] bytes) {
        StringBuffer stringBuffer = new StringBuffer();
        for (int i = 0; i < bytes.length; i++) {
            stringBuffer.append((char) bytes[i]);
        }
        return stringBuffer.toString();
    }
```

Este proceso recibe como parámetro la variable del tipo byte, "*plainText*", que contiene los datos des encriptados. El ciclo contenido dentro de este proceso se encarga de ir tomando los bytes de "*plainText*" para almacenarlos en una variable de tipo *StringBuffer*, el ciclo concluye al haber traspasado todos los bytes. Por último la variable buffer, "*stringBuffer*", se transforma a una de tipo string denominada "*decryptedText*" la cual es finalmente retornada para trabajar con la misma.

Anexo 26. Clase *Temperatura* (Aplicación web).

```
package axisproy.clases;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Coordenadas
 */
public class Temperatura extends HttpServlet {
    private static final long serialVersionUID = 1L;
    static Connection conexion = null;
    static ResultSet rs = null;
    static Statement s = null;
    public static void conecta()
    {{
        try
        {{
           Class.forName("com.mysql.jdbc.Driver");
        }
        catch (Exception e)
        {{
           e.printStackTrace();
        }
        try
        {{
            conexion = DriverManager.getConnection
("jdbc:mysql://localhost/axis","root", "holahola");
        }
        catch (SQLException e)
        {{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    public static void consulta(String id)
    {{
        try
        {{
            s = conexion.createStatement();
        }
        catch (SQLException e)
        {{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        try
        {{
             rs = s.executeQuery ("select * from camion"+id+"temp");
        }
        catch (SQLException e)
        {{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

```
    }

    public static void desconecta ()
    {}{
        try
        {}{
            conexion.close();
        }
        catch (SQLException e)
        {}{
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    /**
     * @see HttpServlet#HttpServlet()
     */


    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/plain");
        String id=request.getParameter("id");
        PrintWriter out = response.getWriter();
        conecta();
        consulta(id);
        String cad="";
        try {
            while (rs.next()){
                cad = ((cad)+(rs.getString(2)+"," +rs.getString(3)+","
+rs.getString(4)+",;"));
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        Encripta en = new Encripta();
        String caden = en.encrypt(cad);
        out.write(caden);
        desconecta();
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request,response);
    }
}
```

El código del método "*doGet*" descrito en *Servlet* representado en la figura anterior es igual al definido para el mismo en el *Servlet* "*Coordenadas*". La diferencia del *Servlet* radica en la tabla consultada por el método "*consulta*", ya que el mismo asignará a la variable "*rs*" de tipo *ResultSet* los datos contenidos en la tabla "*camion**X**temp*" (siendo X el identificador del camión). La misma se hace modificando el parámetro de la función "*executeQuery*" con el siguiente código:

rs = s.executeQuery ("select * from **camion"+id+"temp**");