ESCUELA POLITÉCNICA DEL EJÉRCITO

FACULTAD DE INGENIERÍA ELECTRÓNICA

PROYECTO DE GRADO PARA LA OBTENCIÓN DEL TÍTULO EN INGENIERÍA ELECTRÓNICA

"DESARROLLO DE UN SISTEMA DE ENTRETENIMIENTO PARA UN TELÉFONO MÓVIL, BASADO EN LA PLATAFORMA J2ME"

DALTON EDUARDO GORDILLO LARCO

SANGOLQUÍ – ECUADOR

SEPTIEMBRE 2005

CERTIFICACIÓN

Certificamos que el presente proyecto de grado titulado “Desarrollo de un sistema de entretenimiento para un teléfono móvil, basado en la Plataforma J2ME”, fue realizado en su totalidad por el señor Dalton Eduardo Gordillo Larco, portador de la cédula de identidad No. 171548956-1, y bajo nuestra dirección.

Ing. Darío Duque	Ing. Byron Navas
DIRECTOR	CODIRECTOR

AGRADECIMIENTO

Agradezco a Dios, por darme salud y vida para poder cumplir con uno de los objetivos que me he trazado en mi vida profesional y personal, la obtención de mi titulo de Ingeniero. También, por haberme dado consuelo y fuerzas en los momentos difíciles, que no fueron pocos, en lo referente a mi vida personal durante mis años de estudio.

Agradezco especialmente a mi abuelito, el Ing. Joaquín Larco Cruz (†), por haberme enseñado muchas cosas de la vida y por haber inspirado la consecución de este título. A él lo considero mi mentor.

Agradezco a mi tío, el Econ. Arnulfo Gordillo Carrera (†), por haberme apoyado de diferentes formas durante mi vida y por haber creído y confiado en mis capacidades.

Agradezco a mis padres, familiares, amigos y compañeros quienes me ayudaron y apoyaron durante toda mi vida estudiantil.

Agradezco a los ingenieros Darío Duque y Byron Navas, por haberme orientado y apoyado durante el desarrollo de este proyecto.

Dalton Gordillo L.

DEDICATORIA

Dedico la realización del presente proyecto a mi mentor, el Ing. Joaquín Larco Cruz (†), quien lastimosamente no está más con nosotros, pero yo sé, que desde el lugar donde él se encuentre, se va sentir muy contento y orgulloso con este logro.

Dalton Gordillo L.

PRÓLOGO

En la actualidad la telefonía inalámbrica móvil está desplazando a la telefonía fija, debido a sus ventajas que tiene ante esta última. Además de la movilidad que nos brinda la telefonía móvil, tenemos otras ventajas, las mismas que encontramos en los terminales y en el servicio que las operadoras ofrecen a sus clientes. Una de las ventajas más importante de la telefonía móvil, es que el tamaño de sus terminales es bastante reducido e incorpora varias funcionalidades. Por otro lado, las operadoras celulares ofrecen una extensa gama de servicios, como son: mensajes de texto, voz y multimedia, conexión de datos inalámbricos, Internet, descarga de aplicaciones, etc.

Hoy en día, los usuarios y la sociedad que lo rodea, requieren de estos y otros servicios más complejos. Con motivo de brindar y dar soporte a estos y nuevos servicios que brinda la telefonía celular, se hace necesario el desarrollo de aplicaciones que permitan el acceso y la gestión hacia los mismos. Las mismas que residirán en los terminales y también en los equipos que realizan la gestión por parte de las operadoras.

Por lo expuesto anteriormente, la realización del presente proyecto, “Desarrollo de un sistema de entretenimiento para un teléfono móvil, basado en la Plataforma J2ME”, busca proveer de un nuevo servicio de entretenimiento para los usuarios de telefonía móvil inalámbrica. Este servicio puede ser ofertado por el operador, el fabricante del equipo celular o por el desarrollador del sistema, pero necesariamente debe ser gestionado en conjunto con el operador de la red.

Dado que la tecnología Java 2 Micro Edition (J2ME) es una de las más extendidas y utilizadas en el desarrollo de aplicaciones para teléfonos celulares, uno de los objetivos que se persigue con la realización de este proyecto, es la creación de una referencia para el futuro desarrollo de otras aplicaciones basadas en esta tecnología. Cabe indicar que, en la

actualidad existen en el mercado otras tecnologías que permiten el desarrollo de estas aplicaciones, de manera que J2ME no es la única opción.

ÍNDICE GENERAL

CAPÍTULO I

1. INTRODUCCIÓN

1.1. DESCRIPCIÓN DEL CAPÍTULO

En este capítulo se realiza una introducción a la arquitectura, configuración y perfil de aplicaciones utilizadas en los sistemas inalámbricos móviles, basadas en la tecnología Java, específicamente en la tecnología Java 2 Micro Edition.

Dado que el producto final de este proyecto se basa en la comunicación entre el móvil y el servidor de información, dentro de este capítulo se incluye una descripción de este proceso, así como también de las entidades que forman parte del mismo. Esta descripción se la realiza asumiendo que el proveedor de servicios de telefonía celular utiliza la tecnología CDMA2000 1xRTT.

1.2. DISPOSITIVOS DE INFORMACIÓN Y LA REVOLUCIÓN INALÁMBRICA

Los dispositivos de información con capacidades de conexión, personalizados e inteligentes se están volviendo cada vez más importantes en nuestra vida profesional y personal. Estos dispositivos, entre los que se encuentran los teléfonos móviles, pagers, screen phones y terminales POS, tienen muchas cosas en común; pero también difieren en sus características, forma y funciones. En conclusión, estos nuevos dispositivos tienen como objetivo cumplir funciones más generales, no como sus antecesores que estaban pensados para cumplir funciones específicas y limitadas.

El número de estos dispositivos de información está creciendo rápidamente. En lo que concierne al desarrollo de este proyecto, un indicador importante representa el número de

usuarios de telefonía celular, el mismo que fue de 1,802[1] billones al mes de marzo de este año en el mundo entero.

Por este y algunos otros indicadores se espera que en los próximos años, la mayoría de estos nuevos dispositivos estén conectados al Internet. Esto ocasionará un cambio radical en la forma en que la gente percibe el uso de estos dispositivos de información. Los usuarios de estos dispositivos desearán tener acceso a la información (Contenido Web, información corporativa y personal) de una manera conveniente en cualquier lugar, a cualquier hora y desde una variedad de dispositivos.

Por todo esto, el desarrollo de aplicaciones para los dispositivos inalámbricos, se torna parte esencial en el proceso de ofertar más y mejores servicios. Lo cual no solamente compete a los fabricantes de estos dispositivos, sino también a las empresas proveedoras de servicios. De ahí que, la realización de este proyecto busca ofertar una solución para un campo determinado de las comunicaciones, el mismo que es el entretenimiento.

1.3. EDICIONES JAVA 2

Sun Microsystems lanzó a mediados de los años 90 el lenguaje de programación **Java** que, aunque en un principio fue diseñado para generar aplicaciones que controlaran electrodomésticos, debido a su gran robustez e independencia de la plataforma donde se ejecutase el código, desde sus comienzos fue utilizado para la creación de componentes interactivos integrados en páginas Web y programación de aplicaciones independientes. Estos componentes se denominaron **applets**. Con los años, Java ha progresado enormemente en varios ámbitos como servicios HTTP, servidores de aplicaciones y acceso a bases de datos (JDBC). Como se puede apreciar, Java se ha ido adaptando a las necesidades tanto de los usuarios como de las empresas, ofreciendo soluciones y servicios tanto a unos como a otros. Debido a la explosión tecnológica de estos últimos años, Java ha desarrollado soluciones personalizadas para cada ámbito tecnológico. Sun ha agrupado cada uno de esos ámbitos en una edición distinta de su lenguaje Java, las mismas que son descritas a continuación:

[1] Fuente: Informa Telecoms & Media, WCIS, Marzo 2005

- ***Java 2 Standard Edition (J2SE).*** Orientada al desarrollo de aplicaciones para el entorno personal. Esta edición es la que en cierta forma recoge la iniciativa original del lenguaje Java y tiene las siguientes características:

    - Inspirado inicialmente en C++, pero con componentes de alto nivel, como soporte nativo de Strings y recolector de basura.
    - Código independiente de la plataforma, precompilado a bytecodes intermedio y ejecutado en el cliente por una JVM (Java Virtual Machine).
    - Modelo de seguridad tipo *sandbox* proporcionado por la JVM.
    - Abstracción del sistema operativo subyacente mediante un juego completo de APIs de programación.

Esta versión de Java contiene el conjunto básico de herramientas utilizadas para desarrollar Java Applets, así como las APIs orientadas a la programación de aplicaciones de usuario final, como lo son: interfaz gráfica de usuario, multimedia, redes de comunicación, etc.

- ***Java 2 Enterprise Edition (J2EE).*** Esta edición representa un superconjunto de J2SE, pensada para cubrir las necesidades del entorno corporativo con énfasis en el desarrollo del lado del servidor, utilizando Enterprise JavaBeans (EJBs), aplicaciones web (servlets y páginas JavaServer), CORBA y XML.

- ***Java 2 Micro Edition (J2ME).*** Esta edición de Java está enfocada al desarrollo de aplicaciones para dispositivos electrónicos de información con capacidad de procesamiento, almacenamiento y presentación gráfica muy reducida, como por ejemplo teléfonos móviles, PDAs o electrodomésticos inteligentes. J2ME tiene unos componentes básicos que la diferencian de las otras versiones, una de ellas es el uso de una máquina virtual denominada KVM (Kilo Virtual Machine) en vez de utilizar la JVM clásica.

La Figura 1.1 ilustra la arquitectura de la tecnología Java 2.

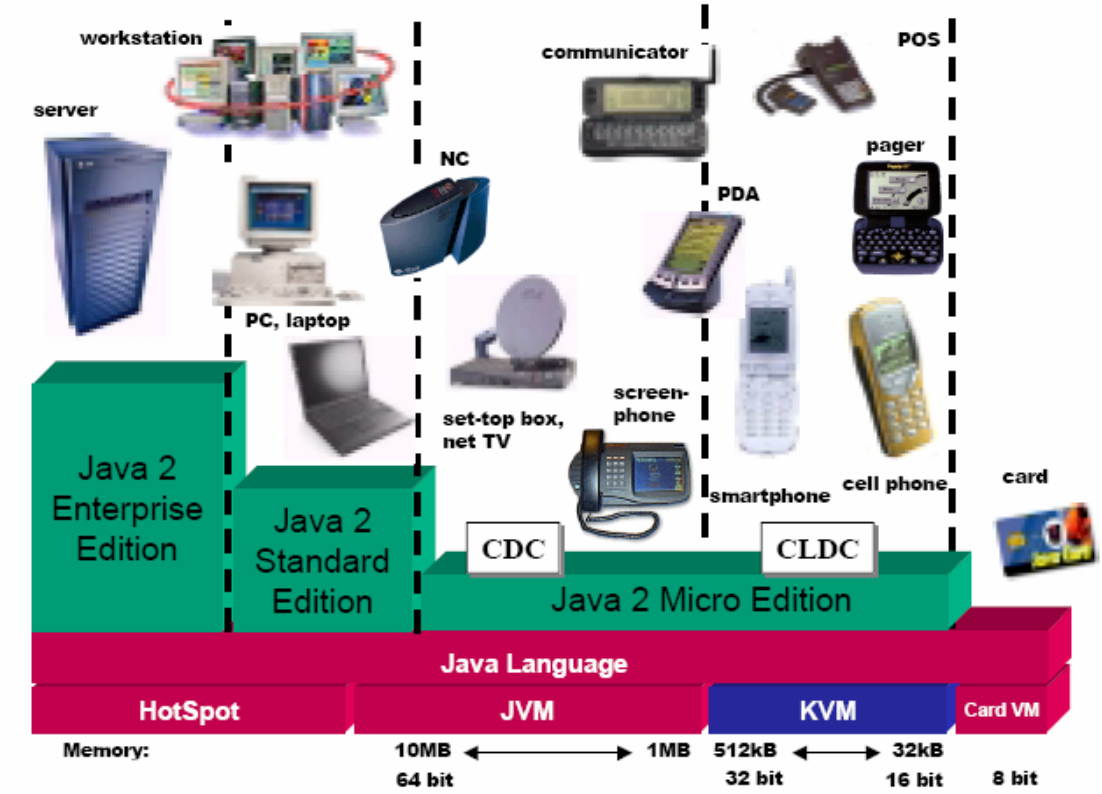

Figura. 1.1. Arquitectura de la tecnología Java 2 y el segmento de mercado que soportan

En la actualidad la tecnología Java puede ser vista como un conjunto de tecnologías que abarca todos los ámbitos de la computación, con dos elementos en común:

- El código fuente del lenguaje Java es compilado a código intermedio interpretado por una máquina virtual Java (JVM), por lo que el código ya compilado es independiente de la plataforma.
- Todas las tecnologías comparten un conjunto más o menos amplio de APIs básicas del lenguaje, agrupadas principalmente en los paquetes **java.lang** y **java.io**.

Este último punto se puede evidenciar en el hecho de que J2ME contiene una mínima parte de las APIs de Java, debido a que la edición estándar de APIs ocupa 20 MB y los dispositivos de información disponen de una cantidad de memoria reducida. En concreto J2ME utiliza 37 clases de la Plataforma J2SE provenientes de los paquetes **java.lang**, **java.io** y **java.util**.

Como se puede ver, J2ME representa una versión simplificada de J2SE. Sun separó estas dos versiones de Java, por cuanto J2ME estaba pensada para dispositivos con limitaciones de procesamiento y capacidad gráfica. También separó J2SE de J2EE, porque este último exigía unas características muy pesadas o especializadas de entrada/salida de información, trabajo en red, etc. En conclusión, se puede considerar a J2ME y J2EE como una versión reducida y ampliada de J2SE, respectivamente (Figura 1.2).

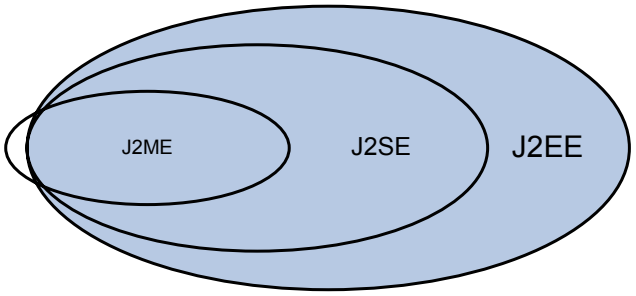

Figura. 1.2. Relación entre las tecnologías Java 2

Cabe anotar que J2ME es la solución que Sun Microsystems provee para el desarrollo de aplicaciones utilizadas en dispositivos móviles. Por otro lado, existen otros lenguajes que permiten desarrollar este tipo de aplicaciones, como son: C++ y Visual Basic/. NET.

1.4. JAVA 2 MICRO EDITION (J2ME)

1.4.1. Modelo de Software, Configuraciones y Perfiles de J2ME

Con el fin de que esta Plataforma sea flexible y se pueda adaptar a los requerimientos de desarrollo que demanda el mercado, la arquitectura J2ME está diseñada para ser modular y escalable. Estas características están definidas por J2ME como tres capas de software constituidas sobre el Sistema Operativo del dispositivo y que se describe a continuación:

- ***Máquina Virtual Java (JVM).*** Esta capa es una implementación de la máquina virtual Java, adaptada a las necesidades del sistema operativo residente en el dispositivo y soporta una configuración J2ME en particular. J2ME utiliza KVM, sobre la cual se realiza una descripción más adelante.

- ***Configuración.*** Define el conjunto mínimo de características que debe tener la máquina virtual Java, así como de librerías Java que deben estar disponibles en una "categoría" particular de dispositivos, representando un segmento horizontal del mercado. En otras palabras, define el "mínimo común denominador" de características y librerías de la Plataforma Java, que los desarrolladores pueden asumir que están disponibles en todos los dispositivos. Esta capa es menos visible para los usuarios, pero es muy importante para los encargados de implementar los perfiles. Existen dos tipos de configuraciones en J2ME: **Connected Limited Device Configuration (CLDC)** enfocada a dispositivos con restricciones de procesamiento y memoria y **Connected Device Configuration (CDC)** enfocada a dispositivos con más recursos disponibles.
- ***Perfil.*** Define el conjunto mínimo de APIs (Application Programming Interfaces) disponibles en una familia de dispositivos en particular. Los Perfiles son implementados en base a una "configuración" específica. Las aplicaciones son desarrolladas para un perfil en particular y por ende pueden ser utilizadas en cualquier dispositivo que soporte ese perfil. Un dispositivo puede soportar múltiples perfiles. Esta capa es la más visible para los usuarios y para los proveedores de las aplicaciones.

En la Figura 1.3 se ilustra el modelo de capas descrito más arriba.

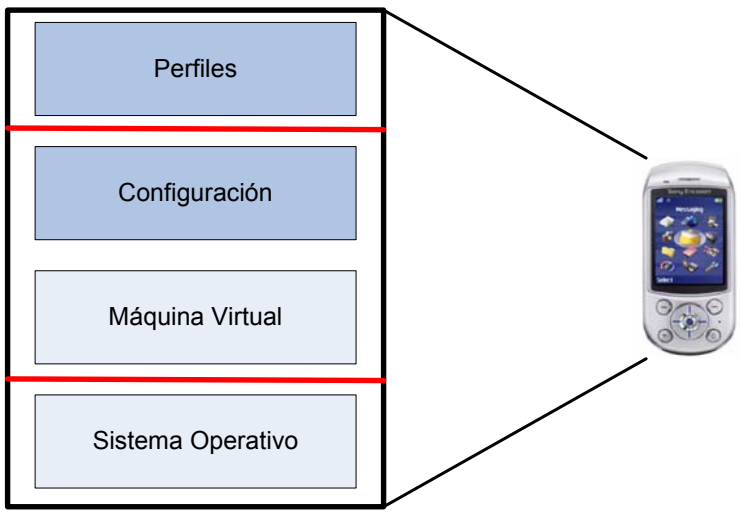

Figura. 1.3. Modelo de software en capas de la Plataforma J2ME

A continuación se realiza una descripción un poco más profunda sobre cada una de estas capas de software.

1.4.2. Máquina Virtual Java (JVM)

Una JVM es un programa encargado de interpretar código intermedio (bytecode) de los programas Java precompilados a código máquina ejecutable por la plataforma, efectuar las llamadas pertinentes al sistema operativo subyacente y observar las reglas de seguridad y corrección de código definidas para el lenguaje Java. De esta forma, la JVM proporciona al programa Java independencia de la plataforma con respecto al hardware y al sistema operativo subyacente. Las implementaciones tradicionales de JVM son, en general, muy pesadas en cuanto a la utilización de memoria y capacidad de procesamiento. J2ME define varias JVMs de referencia adecuadas al ámbito de los dispositivos electrónicos que, en algunos casos, suprimen algunas características con el fin de obtener una implementación menos exigente.

Dado que existen dos tipos de configuraciones dentro de J2ME, CLDC y CDC, cada una requiere una VM diferente. La CLDC utiliza la máquina virtual denominada KVM (Kilo Virtual Machine), mientras que la que utiliza la CDC es CVM (Compact Virtual Machine). En esta sección se describirá la KVM, por cuanto es la máquina virtual necesaria para la configuración CLDC, la misma que fue utilizada en el desarrollo de este proyecto.

1.4.2.1. KVM (Kilo Virtual Machine)

La KVM es una máquina virtual Java, compacta y portable, diseñada para dispositivos que tienen recursos limitados. La meta más alta de diseño de la KVM, fue crear una máquina virtual Java lo más completa y pequeña posible y que pudiera mantener todos los aspectos centrales del lenguaje de programación Java. Pero que pudiera ser ejecutada en un dispositivo con recursos limitados y con unos pocos cientos de kilobytes disponibles en memoria.

Específicamente la KVM fue diseñada para cumplir con los siguientes lineamientos:

- Pequeña, con una ocupación de memoria estática por parte del núcleo de la máquina virtual en el rango de 40 kB a 80 kB (dependiendo de la plataforma y las opciones de compilación).
- Limpia, bien comentada y altamente portable.
- Modular.
- Lo más completa y rápida posible sin sacrificar las características de diseño.

La "K" en KVM representa "kilo" y fue nombrada así porque la capacidad de memoria utilizada está en el rango de los kilobytes. KVM está pensada para microprocesadores de 16/32 bits RISC/CISC con una capacidad de memoria menor a unos cientos de kilobytes. Esto aplica típicamente para teléfonos celulares digitales, pagers y organizadores personales.

La capacidad mínima de memoria requerida para la implementación de una KVM es aproximadamente de 128 kB; incluyendo la máquina virtual, el mínimo conjunto de librerías Java especificadas por la configuración y una cierta cantidad de "heap space" para correr las aplicaciones Java. Una implementación más típica de KVM, requiere de 256 kB de capacidad en memoria, de donde la mitad de ella es utilizada como "heap space" para las aplicaciones, de 40 a 80 kB son necesarios para la máquina virtual en si y el resto es reservado para las librerías.

En la actualidad, la KVM y la CLDC están íntimamente relacionadas. La CLDC se ejecuta solamente sobre la KVM y es la única configuración que esta VM (Virtual Machine) puede soportar. Sin embargo, con el pasar del tiempo se espera que la CLDC se ejecute sobre otra implementación de máquina virtual J2ME y que la KVM quizá pueda soportar otro tipo de configuraciones.

1.4.3. Configuración

Existen dos tipos de configuraciones en la plataforma J2ME:

1.4.3.1. Connected Device Configuration (CDC)

Esta configuración está orientada a dispositivos con una capacidad de procesamiento y de memoria considerable, como por ejemplo decodificadores de televisión digital, televisores con Internet, algunos electrodomésticos inteligentes y sistemas de navegación en automóviles. La máquina virtual de CDC tiene similares características a la utilizada por la Edición Java2SE, pero con limitaciones en lo referente a la capacidad de visualización y de memoria del dispositivo. Esta máquina virtual se denomina **Compact Virtual Machine (CVM)**. Los dispositivos que utilizan la CDC deben cumplir con las siguientes características:

- Microprocesadores de 32 bits.
- 2 MB de memoria, incluyendo memoria RAM y ROM.
- Poseer la funcionalidad completa de la Máquina Virtual Java2.
- Conectividad de red.

La CDC está basada en J2SE ver. 1.3 e incluye paquetes Java de esta edición. Existen algunas particularidades sobre la configuración CDC, las mismas que se encuentran principalmente en el paquete **javax.microedition.io**. Este paquete incluye soporte para comunicaciones HTTP y otras basadas en datagramas. En la Tabla 1.1 se listan las librerías de esta Configuración.

Paquete	Descripción
java.io	Clases e interfases estándar de entrada/salida.
java.lang	Clases básicas del lenguaje.
java.lang.ref	Clases de referencia.
java.lang.reflect	Clases e interfases de reflection.
java.math	Paquete para soporte de operaciones matemáticas.
java.net	Clases e interfases de red.
java.security	Clases e interfases de seguridad.
java.security.cert	Clases de certificados de seguridad.
java.text	Paquete de texto.

Tabla. 1.1. Librerías de la Configuración CDC

Paquete	Descripción
java.util	Clases de unidades estándar.
java.util.jar	Clases y utilidades para archivos JAR.
java.util.zip	Clases y utilidades para archivos ZIP y comprimidos.
javax.microedition.io	Clases e interfases para conexión genérica CDC.

Librerías de la Configuración CDC (*Continuación de la Tabla 1.1*)

1.4.3.2. Connected Limited Device Configuration (CLDC)

La configuración CLDC está orientada a dispositivos de conexión con capacidades limitadas de procesamiento, visualización gráfica y memoria, como por ejemplo teléfonos móviles, pagers, PDAs, etc. Algunas de estas limitaciones vienen dadas por la máquina virtual que esta configuración utiliza y que es la KVM. Los dispositivos que utilizan la CLDC deben cumplir con las siguientes características:

- 160 a 512 kB de memoria disponible. Como mínimo deben disponer de 128 kB de memoria no volátil para la máquina virtual Java y las librerías CLDC y 32 kB de memoria volátil para la máquina virtual en ejecución.
- Procesador de 16 o 32 bits, con al menos 25 MHz de velocidad.
- Conectividad con redes. Generalmente sin cable, con conexión intermitente y ancho de banda limitado (alrededor de 9600 bps).

La CLDC aporta las siguientes funcionalidades a los dispositivos:

- Subconjunto de librerías de la edición estándar Java.
- Máquina virtual KVM con todas sus restricciones.
- Soporte para entrada/salida de datos.
- Soporte para acceso a redes.
- Seguridad.

La Tabla 1.2 nos muestra las librerías incluidas en la CLDC.

Paquete	Descripción
java.io	Clases e interfases estándar de entrada/salida de datos.
java. lang	Clases básicas del lenguaje.
java.util	Clases, interfases y utilidades estándar.
javax.microedition.io	Clases e interfases de conexión genérica CLDC.

Tabla. 1.2. Librerías de la Configuración CLDC

Un aspecto a tener muy en cuenta es la seguridad en CLDC. Esta configuración posee un modelo de seguridad *sandbox* al igual que ocurre con los applets. De todas formas, cualquiera que sea la Configuración seleccionada, esta no se encarga del mantenimiento del ciclo de vida de la aplicación, interfases de usuario o manejo de eventos, sino que estas responsabilidades recaen sobre los Perfiles.

1.4.4. Perfil (Profile)

Esta capa define el mínimo conjunto de APIs para una familia de dispositivos en particular. Los Perfiles son implementados sobre una Configuración específica. Esto hace que a la hora de construir una aplicación se cuente tanto con las APIs del perfil como con las de la configuración. En conclusión, se puede pensar en un perfil como un conjunto de APIs que dotan a una Configuración de funcionalidad específica.

De los perfiles existentes, algunos se encuentran constituidos sobre la configuración CDC y otros sobre la CLDC. Los perfiles constituidos sobre la CDC son los siguientes:

- Foundation Profile.
- Personal Profile.
- RMI (Remote Method Invocation) Profile.

Mientras que los perfiles constituidos sobre la configuración CLDC son los siguientes:

- PDA (Personal Digital Assistants) Profile.
- Mobile Informaction Device Profile (MIDP).

A continuación, en la Figura 1.4, se presenta el diagrama de Perfiles de la Plataforma J2ME.

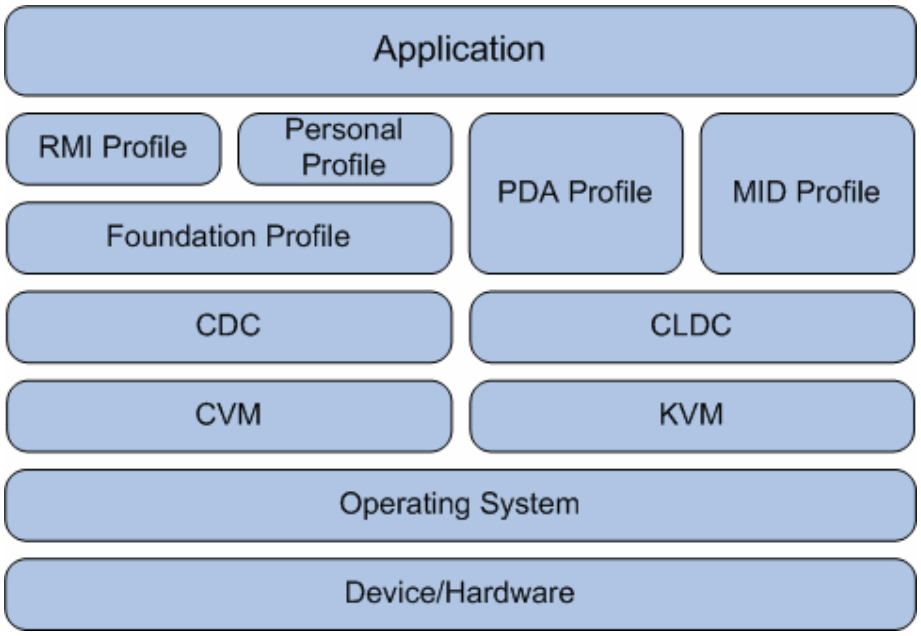

Figura. 1.4. Diagrama de Perfiles de la Plataforma J2ME

Un perfil puede ser construido sobre cualquier otro. A continuación se realiza una breve descripción de cada unos de los perfiles mencionados anteriormente:

1.4.4.1. Foundation Profile

Este perfil define una serie de APIs sobre la CDC orientadas a dispositivos que carecen de interfaz gráfica, como por ejemplo decodificadores de televisión digital. Este perfil incluye gran parte de los paquetes de la J2SE, pero excluye los paquetes **java.awt** y **java.swing**, los mismos que conforman la interfaz gráfica de usuario (GUI) de J2SE.

1.4.4.2. Personal Profile

Es un subconjunto de la plataforma J2SE Ver. 1.3 y proporciona un entorno de programación con un completo soporte gráfico AWT (Abstract Window Toolkit). El

objetivo de este perfil es dotar a la configuración CDC de una interfaz gráfica completa, con capacidades web y que soporte applets. Este perfil requiere de una implementación del Foundation Profile.

1.4.4.3. RMI (Remote Method Invocation) Profile

Este perfil es un subconjunto de las APIs J2SE Ver. 1.3 RMI. Algunas características de estas APIs se han eliminado del perfil RMI debido a las limitaciones de procesamiento y memoria de los dispositivos. Los paquetes eliminados de la versión 1.3 de J2SE RMI son:

- java.rmi.server.disableHTTP
- java.rmi.activation.port
- java.rmi.loader.packagePrefix
- java.rmi.registry.packagePrefix
- java.rmi.server.packagePrefix

1.4.4.4. PDA Profile

Este perfil pretende abarcar PDAs de gama baja, tipo Palm, con una pantalla y algún tipo de puntero y una resolución de al menos 20000 pixeles (200x100). No es posible brindar más información sobre este perfil, porque se encuentra en fase de desarrollo.

1.4.4.5. Mobile Information Device Profile (MIDP)

Este perfil fue el primero en ser definido para la plataforma J2ME y está orientado a dispositivos con las siguientes características:

- Reducida capacidad de procesamiento y de memoria.
- Conectividad limitada (alrededor de 9600 bps).
- 8 kB de memoria no volátil para datos persistentes.
- 32 kB de memoria volátil en tiempo de ejecución para la pila Java.

Los dispositivos que cumplen con estas características son: teléfonos móviles, pagers o PDAs de gama baja con conectividad. El perfil MIDP establece las capacidades del dispositivo, por lo tanto, específica las APIs relacionadas con:

- La aplicación (semántica y control de la aplicación).
- Interfaz de usuario.
- Almacenamiento persistente.
- Trabajo en red.
- Temporizadores.

Las aplicaciones realizadas utilizando MIDP, reciben el nombre de MIDlets. En resumen, un MIDlet es una aplicación Java realizada con el perfil MIDP sobre la configuración CLDC.

1.5. ACCESO DEL MÓVIL A INTERNET EN UNA RED CDMA2000 1xRTT

1.5.1. La Red CDMA2000 1xRTT y sus Componentes

Luego de haber proporcionado una introducción sobre la plataforma J2ME, en esta sección se intenta explicar como se establece la conexión entre el móvil y el servidor mediante una Red CDMA2000 1xRTT, así como las principales entidades que forman parte de este proceso. Para ello a continuación se presenta el diagrama de una red de este tipo de tecnología.

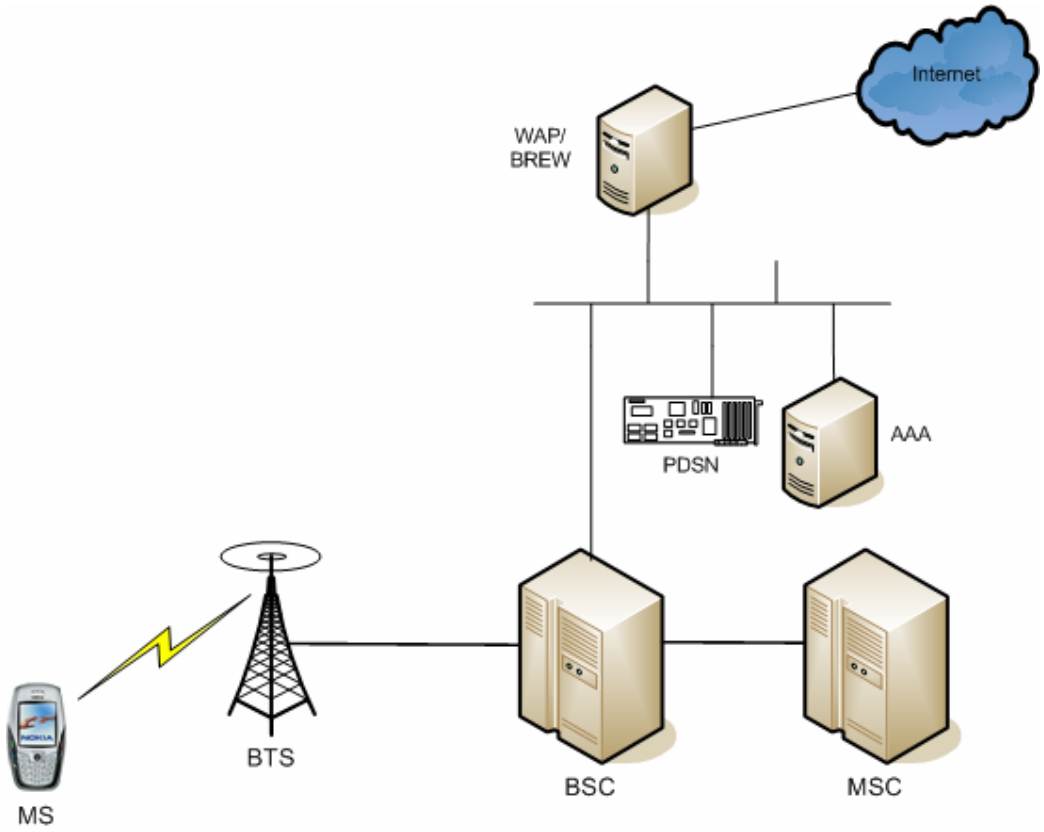

Figura. 1.5. Diagrama de una Red CDMA2000 1xRTT

A continuación se realiza una descripción de cada una de estas entidades:

MS (Mobile Station). Es el terminal móvil que utiliza el usuario para acceder a cualquier servicio ofrecido por el operador de la Red CDMA. Cabe anotar que este dispositivo electrónico debe ser de la misma tecnología que la red; es decir CDMA, y debe soportar el protocolo IS-2000/IS-95A.

BTS (Base Station Transceiver Subsystem). Es el equipo que se instala en el sitio mismo de la celda y sirve para enlazar al usuario con la red CDMA. La BTS provee de la interfase aire común para todos los usuarios, en concordancia con el estándar IS-2000/IS-95A (800 MHz). Este equipo trae las antenas, transmisor, receptor, amplificadores de potencia, unidades de temporización y frecuencia, procesamiento del canal de señal y el hardware de la interfase para soportar la comunicación entre los móviles y el controlador de estaciones base. La BTS puede ser configurada como omni-direccional, de 2 sectores o

de 3 sectores y solamente una de esas configuraciones es necesaria para soportar los canales CDMA que se encuentran alojados dentro de la portadora de 1.25 MHz. La BTS cumple además las siguientes funciones:

- Convierte los paquetes HDLC de voz codificada digitalmente en una señal CDMA modulada para su transmisión al terminal móvil.
- Recibe y convierte esta señal CDMA modulada en paquetes HDLC de voz codificada digitalmente.
- Control de potencia.
- Handoffs suaves.

BSC (Base Station Controller). Provee de mensajería y ruteo de señalización entre ella misma, la MTX, su sistema de administración y la red de BTSs. También provee de codificación y decodificación de voz entre el móvil, a través de la BTS, y el PCM de voz para la MTX. Sólo una BTS es necesaria para el sistema y su normal integración con la MTX. La BSC efectúa funciones de procesamiento de llamadas como:

- Control de potencia.
- Opción de servicios.
- Handoffs suaves.

MSC (Mobile Switching Center). De entre las funciones que cumple esta entidad tenemos las siguientes:

- Procesamiento de llamadas: Se refiere a los recursos de software necesarios para establecer una llamada y proveer de los servicios a los usuarios. De una manera simplista, una llamada telefónica es una conexión entre dos participantes y puede ser móvil o a través de una línea fija.
- Administración de movilidad: Es responsable de suministrar o retirar los recursos necesarios para una llamada realizada por un móvil. Las principales funciones de la administración de movilidad son:
    - Suministrar o retirar recursos de manera consistente en base a un algoritmo.
    - Administración de recursos para el establecimiento de la llamada.

    - Proveer de un interfaz de mensajería para el handoff inter-MTX.
- Facturación.
- Almacenamiento de perfiles de los abonados.
- Soporte de servicios y capacidades.

PDSN (Private Data Serving Node). Es el gateway entre la Red Celular CDMA, en la BSC, y la Red de Datos Pública Conmutada (PSTN). Tiene conectividad con la BSC a través de una interfase ethernet de 10 Mb. Una PDSN puede soportar un área metropolitana de servicios (MTA: Metropolitan Trade Area). Típicamente una MTA puede tener 3 MTXs. Una unidad móvil utiliza un enlace inalámbrico hacia la red celular y la línea fija utiliza una conexión cableada. Con estos atributos, una conexión puede ser clasificada como fija, fija-móvil, móvil-fija o móvil-móvil.

Servidor Radius AAA (Authentication, Authorization and Accounting). Una red IS-2000 requiere de un servidor de autenticación, autorización y facturación para aprobar o rechazar el acceso a la Red de Datos (PDN: Packed Data Network). Este servidor implementa un servicio de acceso remoto para autenticación y facturación de los paquetes de datos.

Servidor WAP/BREW. Es el gateway entre la conexión establecida por el móvil y el contenido disponible en Internet.

Cabe indicar que en esta sección alguno o algunos de los componentes de la Red CDMA2000 1xRTT pudieron haber sido obviados. Esto se debe a que nada más fueron incluidas las entidades más relevantes para el desarrollo de este proyecto.

1.5.2. Descripción de la conexión entre el móvil y la red de datos

Una vez conocidas las entidades de la Red CDMA2000 1xRTT, el proceso que un móvil sigue para acceder al Internet a través de esta red es el siguiente:

- En primera instancia el MS intenta realizar una llama de datos, la misma que es encausada hacia el AAA para su validación. Cabe indicar que esta llamada no es

tratada por la MTX, sino que pasa a través de la BSC y la PDSN hacia el servidor AAA.

- El AAA determina si los datos de usuario (user y password) enviados por el móvil son correctos. Si la respuesta es afirmativa, la PDSN le asigna una IP de entre un rango (pool) de IPs que tiene para este efecto. Esta IP asignada al móvil es temporal y dura el tiempo que demore la llamada de voz, el mismo que representa la conexión del móvil con la red de datos. Por el contrario si los datos son incorrectos, el AAA no permite que la PDSN le asigna una IP y como resultado de esto, el usuario no tendrá acceso a la red.
- Una vez que el usuario es cliente de la red de datos, podrá hacer uso de todos los servicios que su perfil de abonado disponga. De entre estos servicios tenemos: WAP, BREW, acceso a bases de datos privadas, etc.
- Cada vez que el usuario solicita información a la red, se establece una llamada de datos y la PDSN le asigna una nueva IP. Cuando la red entrega toda la información solicitada por el móvil, la llamada conjuntamente con la sesión y conexión se cierran.
- Cabe indicar que el proceso de autenticación a través del AAA ocurre solamente al inicio de este proceso.
- Un punto muy importante es que el MIDlet desarrollado en este proyecto podrá ser descargado a través de BREW para luego poder ser instalado en el móvil.
- Las peticiones realizadas a la Web por la aplicación residente en el móvil (MIDlet), podrán ser manejadas por el Servidor WAP.

1.6. CONVENCIONES UTILIZADAS EN ESTE PROYECTO

función(parámetros)	Los nombres de funciones van con minúsculas y negrillas.
función(...)	Los tres puntos suspensivos "...", son incluidos en lugar de los parámetros, para indicar que la función lleva más de un parámetro. Se convino esto, por cuanto los parámetros de algunas funciones son extensos.
Clase o "Clase"	Los nombres de las clases van con negrillas o también entre comillas dobles.

Palabra_Clave	Se encuentran con negrillas las palabras clave y los tipos de datos, ambos pertenecientes al lenguaje Java.

CAPÍTULO II

2. MIDLETS

2.1. DESCRIPCIÓN DEL CAPÍTULO

Los MIDlets son aplicaciones desarrolladas en base al perfil MIDP y están diseñados para ser ejecutados en dispositivos móviles inalámbricos de comunicación, como son los teléfonos celulares y PDAs (Personal Digital Assistants). Estos dispositivos tienen poca capacidad de procesamiento, gráfica y de memoria por esta razón los MIDlets residen en un software encargado de ejecutarlos y gestionar todos los recursos que estos necesiten, este software es el **Gestor de Aplicaciones (AMS: Application Management Software)**. Uno o más MIDlets pueden ser agrupados para formar un **MIDlet suite**.

En este capítulo se realiza una descripción sobre el AMS, así como también se incluye información básica sobre los MIDlets (propiedades, ciclo de vida y flujo de estados) y sobre el paquete **javax.microedition.midlet**. Por último se incluye la definición de las interfases gráficas de usuario y dentro de estas, la interfaz de alto nivel.

2.2. GESTOR DE APLICACIONES (AMS: APPLICATION MANAGEMENT SOFTWARE)

Como se mencionó anteriormente, el AMS es el encargado de ejecutar y gestionar el funcionamiento de los MIDlets. Este software reside en el móvil y a través de él se puede ejecutar, pausar o destruir las aplicaciones J2ME. El AMS desempeña dos funciones:

1. Gestionar el ciclo de vida de los MIDlets.
2. Controlar los estados por los que pasa el MIDlet mientras está en ejecución.

2.2.1. Ciclo de vida de un MIDlet

El ciclo de vida del MIDlet consta de 5 fases como se lo puede apreciar en la Figura 2.1.

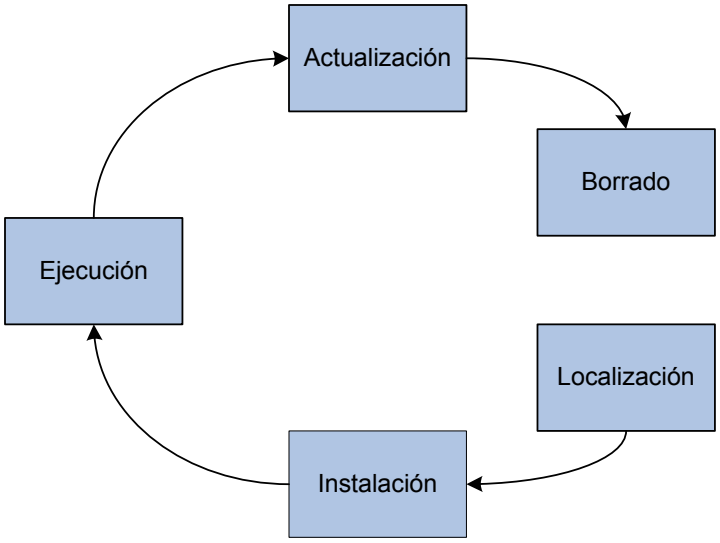

Figura. 2.1. Ciclo de vida de un MIDlet

Como se mencionó anteriormente, el AMS gestiona el ciclo de vida de un MIDlet. A continuación se realiza una descripción de cada una de las fases de este ciclo de vida:

Localización. En esta fase el AMS nos permite seleccionar dentro de todos los MIDlets disponibles, el que deseemos descargar. La operación de descarga de la aplicación también se encuentra contemplada dentro de esta fase y depende de las capacidades que tenga el dispositivo que estemos utilizando. Algunas de las formas de llevar a cabo esta operación son: a través de un cable conectado a la PC o mediante una conexión inalámbrica.

Instalación. Una vez descargada la aplicación comienza la fase de instalación, en el transcurso de la cual el AMS se encarga de informar al usuario sobre el progreso de la operación, así como cualquier inconveniente que se pudiere presentar durante la misma. Una vez instalada la aplicación, todas sus clases, archivos y almacenamiento persistente se encuentran listos para su uso.

Ejecución. En esta fase el AMS permite iniciar la ejecución de la aplicación (MIDlet). Luego de lo cual se encargará de controlar y gestionar los estados del MIDlet en función de los eventos que se produzcan. El tema de los estados del MIDlet se lo tratará más adelante en detalle.

Actualización. El AMS tiene la capacidad de detectar si un MIDlet descargado es una versión actualizada de uno anterior; es decir, si una versión anterior de esta aplicación ya se encuentra almacenada en el dispositivo o no. De ser así, nos debe informar sobre este particular y darnos la alternativa de realizar la actualización.

Borrado. En esta fase el AMS borra el MIDlet seleccionado, previamente nos presenta un mensaje de confirmación, así como también cualquier error que se presente durante esta operación.

Una vez instalado el MIDlet en el móvil, queda almacenado permanentemente en una zona de almacenamiento persistente del dispositivo. El usuario es el que decide en qué momento proceder a eliminarlo y en ese caso el AMS es el encargado de gestionarlo, así como de presentar cualquier error si se llegara a suscitar.

2.2.2. Estados de un MIDlet en fase de ejecución

El MIDlet durante su ejecución puede pasar por 3 estados diferentes (Figura 2.2) y que son:

1. **Activo.** En este estado el MIDlet se encuentra en ejecución.
2. **Pausa.** El MIDlet no se encuentra en ejecución. En este estado la aplicación no debe utilizar ningún recurso compartido. Para volver a la ejecución, el MIDlet debe cambiar su estado a “Activo”.
3. **Destruido.** La aplicación no puede pasar a ningún otro estado y todos los recursos que fueron utilizados son liberados. Para regresar a los otros dos estados anteriores, se debe volver a ejecutar la aplicación.

Como se puede apreciar en la Figura 2.2, el MIDlet puede cambiar de estado llamando a los métodos **startApp()**, **pauseApp()** o **destroyApp(...)** a través del AMS. Cabe anotar que el MIDlet puede también cambiar de estado por si solo.

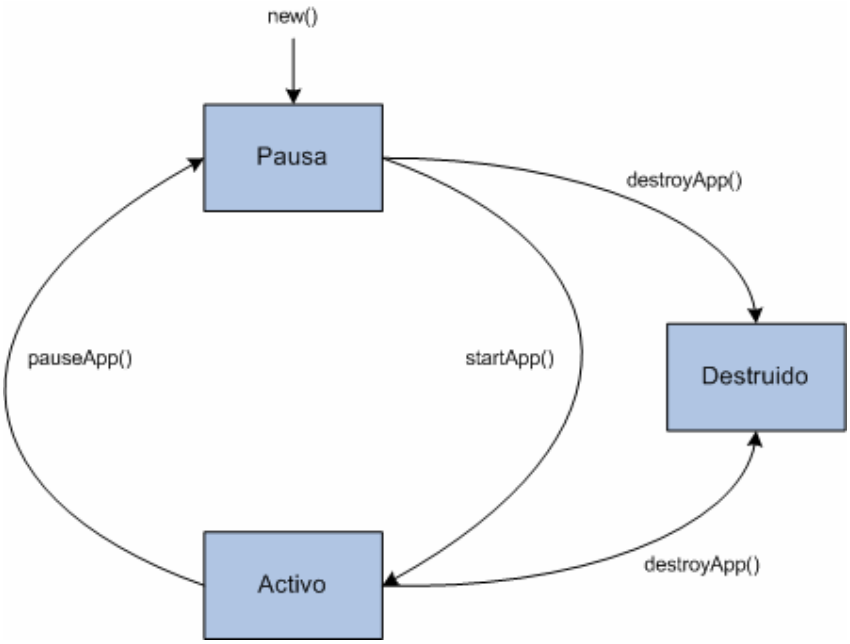

Figura. 2.2. Estados de un MIDlet en fase de ejecución

Cuando un MIDlet se encuentra en ejecución, como primer paso se realiza una llamada a su constructor, para luego pasar al estado de "Pausa" por un instante. El AMS por su parte crea una nueva instancia del MIDlet. Cuando el móvil está listo para ejecutar la aplicación, el AMS invoca al método **startApp()** y pasa al estado "Activo". En este estado el MIDlet utiliza todos los recursos necesarios para su ejecución, pero también se tiene la posibilidad de pasar al estado de "Pausa", ya sea por una acción del usuario o por gestión del AMS con objeto de optimizar recursos. Desde los estados "Activo" o de "Pausa" el MIDlet puede pasar al estado "Destruido", para lo cual se tiene que invocar al método **destroyApp(...)**. Se pasa a este estado ya sea porque el MIDlet ha terminado su ejecución o porque otra aplicación de mayor prioridad necesita ser ejecutada, luego de lo cual todos los recursos utilizados por la aplicación serán liberados.

2.2.3. El paquete javax.microedition.midlet

Este paquete define las aplicaciones MIDP y su comportamiento con respecto al entorno de ejecución. Está conformado por dos clases, cuya descripción se presenta en la Tabla 2.1.

Clase	Descripción
MIDlet	Una aplicación MIDP.
MIDletStateChangeException	Indica que el cambio de estado ha fallado.

Tabla. 2.1. Clases que conforman el paquete javax.microedition.midlet

A continuación se realiza una descripción de estas dos clases:

2.2.3.1. La Clase MIDlet

La aplicación debe extender (derivar) de esta clase para permitir al AMS controlar el MIDlet, recuperar propiedades de su archivo descriptor JAD (Java Application Descriptor File), notificar y solicitar cambios. Los métodos de esta clase permiten al AMS crear, iniciar, detener y destruir el MIDlet. Un MIDlet es un conjunto de clases diseñadas para ser ejecutadas y controladas a través de sus interfases.

En la Tabla 2.2 se realiza una descripción de los métodos más relevantes de esta clase para el desarrollo de este proyecto.

Método	Descripción
protected abstract void destroyApp(boolean unconditional)	Le indica al MIDlet que debe terminar su ejecución y entrar en el estado de “Destruido”. Cuando pase a este estado el MIDlet debe liberar cualquier recurso y guardar cualquier dato en la memoria persistente. Este método puede ser llamado desde los estados de “Pausa” y “Activo”.

Tabla. 2.2. Métodos más relevantes de la Clase MIDlet para este proyecto

Método	Descripción
String getAppProperty(String key)	Provee al MIDlet de un mecanismo para recuperar el valor de cualquiera de los campos de propiedades a través del AMS.
void notifyDestroyed()	Es utilizado por el MIDlet para notificar al AMS que ha entrado en el estado de "Destruido".
protected abstract void pauseApp()	Indica al MIDlet que debe entrar en el estado de "Pausa". En este estado el MIDlet debe liberar cualquier recurso compartido y permanecer inoperante. Este método puede ser llamado únicamente cuando el MIDlet se encuentra en el estado "Activo".
protected abstract void startApp()	Indica al MIDlet que se ha entrado en el estado "Activo". En este estado el MIDlet puede retener recursos. Este método puede ser invocado únicamente cuando el MIDlet se encuentra en el estado de "Pausa".

Métodos más relevantes de la Clase MIDlet para este proyecto (*Continuación Tabla 2.2*)

Cabe recalcar que los métodos presentados en la Tabla 2.2 no son todos los que se encuentran definidos en la Clase MIDlet. La definición completa de los métodos de esta clase se los puede encontrar en la Especificación del Perfil MIDP Ver. 2.0.

2.2.3.2. La Clase MIDletStateChangeException

Indica que el cambio de estado del MIDlet ha fallado. Esta excepción es lanzada por el MIDlet en respuesta a una llamada para el cambio de estado en la aplicación a través de la interfaz del MIDlet.

2.3. ESTRUCTURA DE LOS MIDLETS

A continuación se presenta la estructura que comparten todos los MIDlets en lo referente al código. Cabe recalcar que los MIDlets al igual que los "applets" carecen de la función **main()**. De darse el caso de que dicha función existiese, el Gestor de Aplicaciones la ignoraría por completo.

```
package BetProjectMIDlet;
//Importación de los paquetes necesarios
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

//Constructor
public BetProjectMIDlet(){
          //En esta sección se inicializan las variables miembro y también se realizan
           //ciertas operaciones necesarias para la inicialización de los objetos de tipo
          //Displayable, ej: añadir botones a un objeto de tipo Form, etc.
    }

public void startApp() {
          //Aquí se definen las acciones que el MIDlet debe ejecutar cuando se encuentre en
          //el estado "Activo".
    }

public void pauseApp() {
          //Aquí se definen las acciones que el MIDlet debe ejecutar cuando se encuentre en
          //el estado de "Pausa" y es opcional.
    }

public void destroyApp(boolean unconditional) {
          //En esta parte se incluye el código que requiramos que el MIDlet efectúe cuando
          //pase al estado de "Destruido", aquí es donde se liberan los recursos ocupados
          //por el MIDlet, ej: memoria. Este punto también puede ser implementado de manera
          //opcional.
    }
}
```

Luego de haber proporcionado esta introducción a la tecnología de los MIDlets, a continuación se presenta la definición de las interfases gráficas de alto nivel.

2.4. INTERFASES GRÁFICAS DE USUARIO

El perfil MIDP define las APIs necesarias para los componentes de interfaz de usuario, entrada de datos, manejo de eventos, almacenamiento persistente, conexiones, etc. En lo referente a las interfases gráficas de usuario, en el paquete **javax.microedition.lcdui** se encuentran definidas todas las herramientas necesarias para gestionarlas.

Las interfases gráficas de usuario se encuentran divididas en dos grupos, que son los siguientes:

1. **Interfaz de usuario de alto nivel.** Esta interfaz implementa componentes tales como cajas de texto, listas de selección, botones, notificaciones, etc. Estos elementos son implementados por cada dispositivo y la finalidad de utilizar este tipo de interfases es su portabilidad. La apariencia de estos controles depende exclusivamente de las características del dispositivo donde se esté ejecutando la aplicación. Cabe anotar que no se tiene un control total sobre lo que se presenta en la pantalla, al momento de utilizar estas APIs.

2. **Interfaz de usuario de bajo nivel.** Con estas interfases se tiene todo el control de lo que se presenta en la pantalla, así como también se tiene el control sobre eventos de bajo nivel, como por ejemplo el rastreo de la pulsación de una de las teclas, etc. Generalmente estas interfases son utilizadas en el desarrollo de juegos, en donde el control sobre los eventos presentados en pantalla y sobre las acciones del usuario son muy importantes.

El paquete **javax.microedition.lcdui** contiene las clases necesarias para crear estos dos tipos de interfases gráficas. En la Figura 2.3 se puede apreciar la organización jerárquica de estas clases.

Cabe anotar que para el desarrollo del MIDlet de este proyecto se utilizaron únicamente interfases gráficas de alto nivel, las interfases de bajo nivel no se incluyen en esta documentación por cuanto no fueron utilizadas en el desarrollo de esta aplicación.

2.4.1. Interfaz de usuario de alto nivel

En esta sección, en primer lugar se hará una descripción de las clases de mayor jerarquía del paquete **javax.microedition.lcdui** (Display, Displayable y Command), con el fin de obtener una instrucción básica sobre el manejo de estas interfases gráficas. Luego se profundizará en las clases derivadas de la Superclase Screen, ya que todas estas conforman la interfaz de usuario de alto nivel.

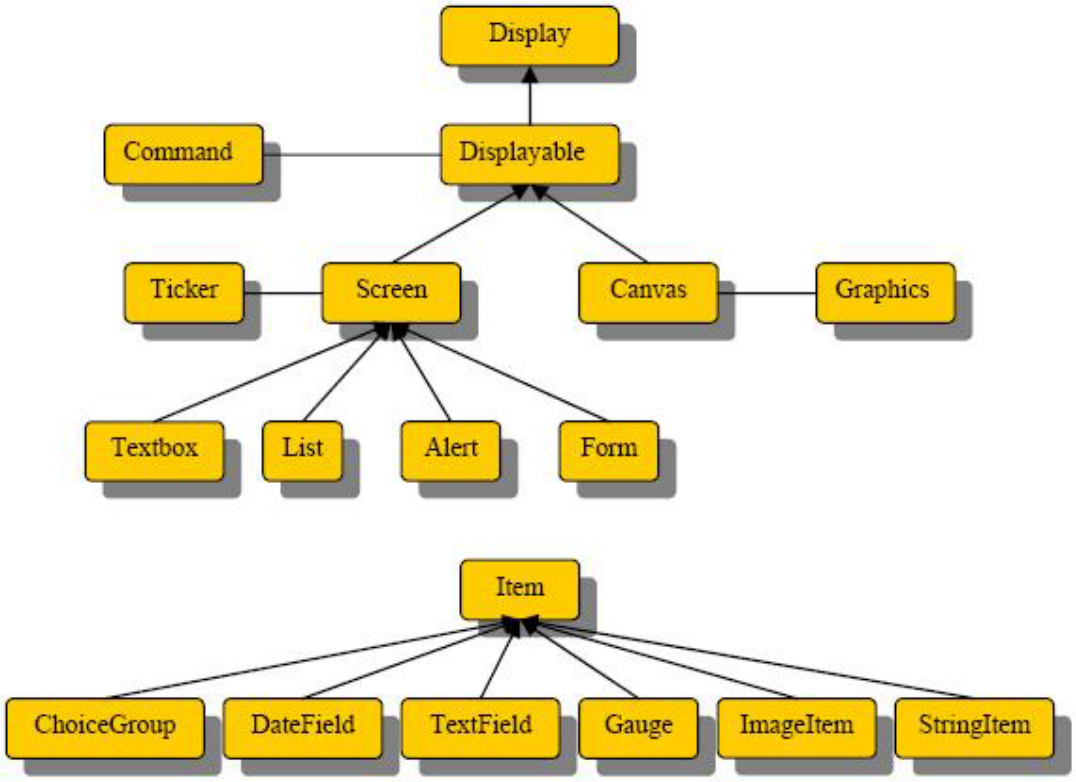

Figura. 2.3. Organización jerárquica de las Clases Display e Item

2.4.1.1. La Clase Display

Esta clase representa el administrador del display y de los dispositivos de entrada del sistema. Todo MIDlet debe poseer al menos un objeto del tipo Display. En este objeto se pueden incluir todos los objetos de tipo Displayable que sean necesarios. Esta clase puede obtener información acerca de las características de la pantalla del dispositivo donde se esté ejecutando el MIDlet, además de ser capaz de mostrar los objetos que componen nuestras interfases. En la Tabla 2.3 se listan algunos de los métodos de esta clase.

Método	Descripción
Displayable getCurrent()	Obtiene el objeto Displayable actual para este MIDlet.
static Display getDisplay(MIDlet m)	Obtiene el objeto Display que es único para este MIDlet.

Tabla. 2.3. Métodos más relevantes de la Clase Display para este proyecto

Método	Descripción
void setCurrent(Alert alert, Displayable nextDisplayable)	Establece la pantalla **nextDisplayable** luego de que el **Alert** haya sido cesado.
void setCurrent(Displayable nextDisplayable)	Establece la pantalla actual.
void setCurrentItem(Item item)	Establece la pantalla en la zona donde se encuentre el **item**.

Métodos más relevantes de la Clase Display para este proyecto (*Continuación Tabla 2.3*)

Para obtener una instancia del objeto Display se utiliza la siguiente sentencia de código:

```
Display myDisplay = Display.getDisplay(this);
```

La llamada a este método se lo debe realizar dentro del constructor del MIDlet, de tal manera que esté a disposición en toda la ejecución de este. Dentro del método **startApp()** se tiene que hacer referencia al objeto Displayable que se necesita que se encuentre activo, para lo cual se utiliza el método **setCurrent(...)**.

2.4.1.2. La Clase Displayable

Representa a los objetos que tienen la capacidad de ser incluidos en el Display (pantalla); es decir, los objetos que conforman o representan las pantallas de nuestra aplicación, la misma que puede contener tantas como sea necesario. Un objeto Displayable puede tener un título, un **ticker**[2], ninguno o algunos comandos, y un **listener**[3] asociado con ellos.

A menos de que no se haya especificado algo a través de las subclases de Displayable, el estado de los objetos de esta clase que son recién creados, tienen las siguientes características por defecto:

- El objeto no es visible en el Display (pantalla).
- No existe ningún ticker asociado con el Displayable.

[2] Un **Ticker** es un segmento de texto que se despliega continuamente a lo largo de la pantalla.
[3] Un **Listener** fija la atención sobre cualquier evento que pudiera ser suscitado sobre los objetos de tipo Command o Item del Displayable donde se encuentran incluidos estos objetos.

- El título es nulo.
- No tiene objetos de tipo Command (comandos) presentes.
- No tiene ningún Listener asociado a estos comandos.

Algunos de los métodos de la Clase Displayable que fueron más relevantes en el desarrollo de este proyecto se definen en la Tabla 2.4.

Método	Descripción
void addCommand(Command cmd)	Añade un comando a la pantalla.
void removeCommand(Command cmd)	Remueve un comando de la pantalla.
void setCommandListener(CommandListener l)	Establece un "Listener" para los comandos de esta pantalla, reemplazando a cualquier "Listener" previo.
void setTitle(String s)	Fija un título al Displayable.

Tabla. 2.4. Métodos más relevantes de la Clase Displayable para este proyecto

2.4.1.3. La Clase Command y la interfaz CommanListener

Un objeto derivado de la clase **Command** nos sirve para detectar algún evento y posteriormente ejecutar una determinada acción. Se puede hacer una analogía de estos objetos con los botones que implementa una aplicación para Windows.

En el constructor de este objeto se deben definir tres parámetros:

1. **Etiqueta.** Es una cadena de texto que contiene el nombre con el cual el comando (botón) aparecerá en la pantalla.
2. **Tipo.** Indica el tipo de comando que se necesita crear. En la Tabla 2.5 se listan los tipos que se pueden asignar a un objeto de tipo Command. La declaración de este tipo sirve para que el dispositivo identifique el Command y le dé una apariencia acorde con el resto de aplicaciones existentes en el mismo.
3. **Prioridad.** Se tiene la posibilidad de fijar una prioridad a estos objetos. Con este dato el AMS puede establecer el orden de aparición de los comandos en el Displayable. A mayor número, menor será la prioridad del comando.

Tipo	Descripción
BACK	Petición para retornar a la pantalla anterior.
CANCEL	Petición para cancelar una operación en curso.
EXIT	Permite salir de la aplicación.
HELP	Muestra información de ayuda sobre la utilización de la aplicación.
ITEM	Petición para ejecutar una acción sobre un *item* del Displayable.
OK	Aceptación de una acción por parte del usuario.
SCREEN	Utilizado en comandos de propósito más general.
STOP	Petición para detener una operación.

Tabla. 2.5. Tipos de apariencia o acción que se pueden asignar a un objeto de tipo Command

En la Tabla 2.6 se listan todos los métodos de la Clase Command.

Método	Descripción
int getCommandType()	Obtiene el tipo de comando (botón).
String getLabel()	Obtiene la etiqueta reducida del comando.
String getLongLabel()	Obtiene la etiqueta extensa del comando.
int getPriority()	Obtiene la prioridad del comando.

Tabla. 2.6. Métodos de la Clase Command

No basta únicamente con crear un objeto de tipo Command para que este ejecute la acción que se especifica en su tipo. Para esto se tiene que implementar la interfaz **CommandListener**, la misma que incorpora un único método que es **void commandAction(Command c, Displayable d)**. En este método es donde se tienen que implementar las acciones u operaciones necesarias para cuando se detecte algún evento sobre el **Command c** presentado en el **Displayable d**.

Luego de haber realizado una introducción a las interfases gráficas de MIDP, en la siguiente sección se va a realizar una descripción de todas y cada una de las subclases de **Screen** (Superclase), ya que estas conforman la interfaz de usuario de alto nivel.

2.4.1.4. La Clase Alert

Un Alert es una pantalla que muestra información al usuario y que se mantiene por un periodo de tiempo definido antes de presentar la siguiente pantalla. Este tipo de objetos pueden contener cadenas de texto y una imagen. El Alert tiene como objetivo, informar al usuario sobre cualquier error u otro tipo de notificación de caracter informativo.

Se pueden definir dos tipos de Alert, los mismos que son:

1. **Modal.** La pantalla de Alert se presenta de manera permanente hasta que el usuario la cancela. Esto se consigue a través del método **setTimeout(...)**.
2. **No Modal.** El Alert permanece activo durante el tiempo que se haya definido en el método **setTimeout(...)**. Una vez que se haya vencido este tiempo, el Alert desaparecerá y a continuación se presenta el Displayable que haya sido definido.

Se puede elegir el tipo de Alert que la aplicación va a presentar y cada uno de estos tiene asociado un sonido. Los tipos de Alert se presentan en la Tabla 2.7.

Método	Descripción
ALARM	Aviso de petición previa.
CONFIRMATION	Indica la confirmación de una acción realizada por el usuario.
ERROR	Indica que ha ocurrido un error.
INFO	Indica algún tipo de información.
WARNING	Indica al usuario que la acción es potencialmente peligrosa.

Tabla. 2.7. Tipos de Alert

Los constructores de esta clase y algunos de sus métodos que fueron más relevantes para el desarrollo de este proyecto se describen en las Tablas 2.8 y 2.9.

Constructor	Descripción
Alert(String title)	Construye un nuevo y vacío objeto Alert con su respectivo título.
Alert(String title, String alertText, Image alertImage, AlertType alertType)	Construye un nuevo objeto Alert con el título, cadena de texto, imagen y tipo de Alert.

Tabla. 2.8. Constructores de la Clase Alert

Método	Descripción
void addCommand(Command cmd)	Añade un botón (Command) al Alert. Cuando la aplicación añade por primera vez un botón, el de Aceptar (**Done**) es removido.
void removeCommand(Command cmd)	Remueve el Command (botón) del Alert. Cuando la aplicación realiza esta acción, el botón de Aceptar (**Done**) es implícitamente añadido.
void setImage(Image img)	Establece la imagen utilizada en el Alert.
void setString(String str)	Establece la cadena de texto utilizada en el Alert.
void setTimeout(int time)	Establece el tiempo que el Alert será presentado.
void setType(AlertType type)	Establece el tipo de Alert.

Tabla. 2.9. Métodos más relevantes de la Clase Alert para este proyecto

2.4.1.5. La Clase List

Esta clase nos permite implementar pantallas en la cuales se pueden incluir una lista de opciones, lo que es muy útil a la hora de trabajar con un menú de opciones. La Clase List implementa la interfaz **Choice**, la misma que hace posible definir uno de los tres tipos de lista que se presentan a continuación:

1. **Exclusiva.** En este tipo de listas, la selección de uno de sus elementos produce que la opción anterior (que estaba seleccionada) quede deseleccionada; es decir, permite la selección de una de sus opciones a la vez. La implementación de este tipo de listas se lo hace a través de **Radio Buttons**. La selección de una de sus opciones no produce ningún efecto, por lo que hay la necesidad de implementar un Command (botón) que permita almacenar el estado de las opciones y determinar cuál de ellas ha sido seleccionada.

2. **Implícita.** Este tipo de listas permite efectuar alguna acción directamente sobre la opción que se haya seleccionada. Este mecanismo de selección constituye el menú clásico de opciones.

3. **Múltiple.** Las listas de este tipo permiten seleccionar todas las opciones deseadas y al igual que las listas "Exclusivas" se necesita un objeto de tipo Command para salvar el contexto.

Los constructores de esta clase y algunos de sus métodos que fueron más relevantes para el desarrollo de este proyecto se describen en las Tablas 2.10 y 2.11.

Constructor	Descripción
List(String title, int listType)	Crea una nueva lista (vacía), especificando su título y tipo.
List(String title, int listType, String[] stringElements, Image[] imageElements)	Crea una nueva lista, especificando su título, tipo y un arreglo de cadenas de texto e imágenes que serán sus contenidos iniciales.

Tabla. 2.10. Constructores de la Clase List

Método	Descripción
int append(String stringPart, Image imagePart)	Añade un elemento a la lista.
void delete(int elementNum)	Remueve el elemento indicado de la lista.
void deleteAll()	Remueve todos los elementos de la lista.
int getSelectedFlags(boolean[] selectedArray_return)	Consulta el estado (de selección) de la lista y retorna el estado de sus elementos en un arreglo de tipo boleano.
int getSelectedIndex()	Retorna el índice de uno de los elementos de la lista que se encuentra seleccionado.
String getString(int elementNum)	Retorna la cadena de texto del elemento indicado.
void insert(int elementNum, String stringPart, Image imagePart)	Inserta un elemento en la lista, justo antes del elemento especificado.
boolean isSelected(int elementNum)	Retorna un valor de tipo boleano, indicando si el elemento está seleccionado.

Tabla. 2.11. Métodos más relevantes de la Clase List para este proyecto

Método	Descripción
void set(int elementNum, String stringPart, Image imagePart)	Establece el String (cadena de texto) y la imagen del elemento indicado, reemplazando a los que estuvieron antes.
int size()	Retorna el número de elementos de la lista.

Métodos más relevantes de la Clase List para este proyecto (*Continuación Tabla 2.11*)

2.4.1.6. La Clase TextBox

La Clase TextBox es una pantalla que permite al usuario ingresar y editar texto. Al momento de crear un objeto de este tipo se debe especificar la capacidad requerida para albergar la información. Esta capacidad puede ser mayor que la del dispositivo, en ese caso el TextBox implementa un mecanismo de *scroll*, para poder desplegar todo el contenido del texto, aunque podría ocurrir que se llegará a sobrepasar su límite. En ese caso se lanzaría una **exception** indicando que se ha excedido la capacidad máxima del objeto. Para controlar y evitar que esto ocurra se puede invocar al método **getMaxSize()**, el mismo que devuelve la capacidad máxima que tiene el objeto para albergar texto.

Los constructores de esta clase, así como los métodos más relevantes para ese proyecto se presentan a continuación.

Constructor	Descripción
TextBox(String title, String text, int maxSize, int constraints)	Crea un nuevo objeto TextBox con el título, contenido inicial, máximo número de caracteres y condiciones indicadas.

Tabla. 2.12. Constructor de la Clase TextBox

Método	Descripción
void delete(int offset, int length)	Borra los caracteres del TextBox.
int getMaxSize()	Retorna el máximo número de caracteres que el TextBox puede alojar.

Tabla. 2.13. Métodos más relevantes de la Clase TextBox para este proyecto

Método	Descripción
int setMaxSize(int maxSize)	Establece el número máximo de caracteres que el TextBox puede contener.
void setString(String text)	Establece el contenido del TextBox como un valor de tipo String (cadena de texto), reemplazando el contenido previo.
void setTitle(String s)	Establece el título del Displayable.
int size()	Retorna el número de caracteres que se encuentran actualmente alojados en el TextBox.

Métodos más relevantes de la Clase TextBox para este proyecto (*Continuación Tabla 2.13*)

2.4.1.7. La Clase Form

Un objeto de tipo Form funciona como un contenedor de un determinado número de objetos de la Clase Item. El número de objetos que pueden ser insertados en el Form es variable, pero dado que las características de los dispositivos MID en cuanto a la capacidad de desplegar formas es reducido, se recomienda que no sean muchos. Cumpliendo este lineamiento se prescinde de la utilización del mecanismo de *scroll.*

Para referirse a los objetos Item incluidos en el Form se utiliza índices, tomando como elemento inicial el 0 y como final el obtenido con el método **size()** menos 1. Por otro lado, cabe señalar que un mismo Item no puede estar en más de un Form a la vez, pero si esto es necesario se debe borrar este Item de la anterior pantalla para luego mostrarla en la actual.

En las Tablas 2.14 y 2.15 se listan los constructores y los métodos más relevantes de esta Clase.

Constructor	Descripción
Form(String title)	Crea un nuevo Form (vacío).
Form(String title, Item[] items)	Crea un nuevo Form con el contenido especificado.

Tabla. 2.14. Constructores de la Clase Form

Método	Descripción
int append(Image img)	Añade un Item, que contiene una imagen, al Form.
int append(Item item)	Añade un Item al Form.
int append(String str)	Añade un Item, que contiene un String, al Form.
void delete(int itemNum)	Borra el Item especificado.
void deleteAll()	Borra todos los objetos Items del Form.
void insert(int itemNum, Item item)	Inserta un Item en el Form, justo antes del objeto especificado.
void set(int itemNum, Item item)	Establece un nuevo Item en la localidad especificada, reemplazando al objeto anterior.
int size()	Retorna el número de Items que tiene el Form.

Tabla. 2.15. Métodos más utilizados de la Clase Form en este proyecto

2.4.1.8. La Clase Item y sus Subclases

Es la superclase que define los objetos que serán incluidos en el Form. No se proporcionará mayor detalla sobre esta Clase sino sobre sus clases descendientes, que son las siguientes:

La Clase TextField

Un objeto de este tipo es un componente de texto editable y que puede ser incluido en un Form. Un TextField puede alojar un máximo número de caracteres y está condición debe ser cumplida al momento de invocar a su constructor, cuando el usuario se encuentra editando su contenido o cuando se llama a algún método para alterar la información que contiene. El número de caracteres presentados y su organización en filas y columnas son determinados por el dispositivo.

TextField comparte el concepto de entrada restringida de datos, como lo tiene también la Clase TextBox. Las diferentes restricciones permiten a la aplicación forzar al usuario para que ingrese los datos de una manera específica, de esa manera el usuario está obligado a ingresar la información en la forma que la aplicación lo requiere, por ejemplo en el caso de que se requiera recolectar información sobre una dirección de e-mail, un password o un número de celular, etc.

Para lograr esto, en la Clase TextField se definen unas constantes que se dividen en dos grupos: el valor de la restricción y la constante **CONSTRAINT_MASK**. Mediante el operador lógico "Y" (&) se puede combinar estas dos constantes para poder obtener el valor de la restricción deseada. El valor de estas constantes y su descripción se pueden encontrar en las Tablas 2.16 y 2.17 respectivamente.

Restricción (Constante)	Descripción
ANY	El usuario puede ingresar cualquier tipo de texto.
EMAILADDR	El usuario puede ingresar una dirección de e-mail.
NUMERIC	Se permite ingresar únicamente valores numéricos enteros.
PHONENUMBER	Se permite ingresar números telefónicos.
URL	Se permite ingresar sólo URLs.
DECIMAL	Se permite el ingreso de valores decimales.

Tabla. 2.16. Valores de la restricción para los objetos TextField

Constante CONSTRAINT_MASK	Descripción
PASSWORD	Indica que el texto que será ingresado es confidencial y en lo posible la información debe ser enmascarada.
UNEDITABLE	Indica que la información no podrá ser editada.
SENSITIVE	Indica que la información que será ingresada es confidencial y no deberá ser almacenada de ninguna manera.
NON_PREDICTIVE	Indica que la información ingresada no tiene relación con la información que se puede encontrar en diccionarios o tablas.
INITIAL_CAPS_WORD	Indica que la primera letra del texto debe ser mayúscula.
INITIAL_CAPS_SENTENCE	Indica que la primera letra de cada oración debe ser mayúscula.

Tabla. 2.17. Constantes CONSTRAINT_MASK

La restricción deseada, representada a través de una de las constantes descritas más arriba o en combinación de estas con cualquier constante **CONSTRAINT_MASK**, es pasada en el parámetro **int constraints** hacia el método constructor de esta Clase.

En las Tablas 2.18 y 2.19 se lista el constructor y los métodos más relevantes de esta Clase.

Constructor	Descripción
TextField(String label, String text, int maxSize, int constraints)	Crea un nuevo TextField con la etiqueta especificada, el contenido inicial, el máximo número de caracteres y con las restricciones indicadas.

Tabla. 2.18. Constructor de la Clase TextField

Método	Descripción
void delete(int offset, int length)	Borra los caracteres del TextField.
String getString()	Recupera el contenido del TextField como un String.
void insert(String src, int position)	Inserta un String en el contexto del TextField.
void setConstraints(int constraints)	Establece las restricciones de entrada del TextField.
void setString(String text)	Establece el nuevo contenido del TextField y el anterior es borrado.
int size()	Recupera el número de caracteres que se encuentran actualmente alojados en el TextField.

Tabla. 2.19. Métodos más utilizados de la Clase TextField en este proyecto

La Clase StringItem

Los objetos de esta clase son cadenas de texto (String), que únicamente despliegan la información; es decir, el usuario no puede interactuar con ella. La etiqueta y el contenido del StringItem pueden ser modificados por la aplicación. La representación visual de la etiqueta puede diferir con respecto al contenido del texto.

En las Tablas 2.20 y 2.21 se listan los constructores y los métodos más relevantes de esta Clase.

Constructor	Descripción
StringItem(String label, String text)	Crea un nuevo objeto de tipo StringItem.
StringItem(String label, String text, int appearanceMode)	Crea un nuevo objeto de tipo StringItem con la etiqueta, el texto y la apariencia especificados.

Tabla. 2.20. Constructor de la Clase StringItem

Método	Descripción
String getText()	Recupera el texto del StringItem o retorna un valor nulo si este se encuentra vacío.
void setText(String text)	Establece el texto del StringItem.

Tabla. 2.21. Métodos más utilizados de la Clase StringItem en este proyecto

La Clase ImageItem

Esta clase nos da la posibilidad de insertar o incluir imágenes en un objeto de tipo Form. Al igual que con los objetos de la Clase StringItem, el usuario no puede interactuar con este objeto (la imagen).

En la Tabla 2.22 se describen los constructores de la Clase ImageItem.

Constructor	Descripción
ImageItem(String label, Image img, int layout, String altText)	Crea un nuevo ImageItem con la etiqueta, imagen, posición (layout) y texto alternativo especificados.
ImageItem(String label, Image image, int layout, String altText, int appearanceMode)	Crea un nuevo ImageItem con la etiqueta, imagen, posición (layout), texto alternativo y apariencia especificados.

Tabla. 2.22. Constructores de la Clase ImageItem

Al momento de llamar a alguno de estos constructores, el parámetro **layout** (posición) permite establecer la posición en la que aparecerá la imagen en la pantalla. Este parámetro puede tomar los valores que se muestran en Tabla 2.23.

Valor	Descripción
LAYOUT_CENTER	Imagen centrada.
LAYOUT_DEFAULT	Posición por defecto.
LAYOUT_LEFT	Imagen posicionada a la izquierda.
LAYOUT_NEWLINE_AFTER	Imagen posicionada tras un salto de línea.
LAYOUT_NEWLINE_BEFORE	Imagen posicionada antes de un salto de línea.

Tabla. 2.23. Valores del parámetro “layout”

Valor	Descripción
LAYOUT_RIGHT	Imagen posicionada a la derecha.

Valores del parámetro "layout" (*Continuación Tabla 2.23*)

El parámetro **altText** del constructor especifica una cadena de texto alternativo que se desplegará en el caso de que la imagen exceda la capacidad de pantalla.

En la Tabla 2.24 se presentan algunos de los métodos que incorpora la Clase ImageItem.

Método	Descripción
void setAltText(String text)	Establece el texto alternativo del ImageItem o un valor nulo si no se provee el texto.
void setImage(Image img)	Establece la imagen contenida dentro del ImageItem.
void setLayout(int layout)	Establece el posicionamiento de la imagen (layout).

Tabla. 2.24. Métodos más utilizados de la Clase ImageItem en este proyecto

Las subclases de la Clase Item, que fueron anteriormente descritas no son todas las que se encuentran definidas para la misma, pero fueron las que se utilizaron en el desarrollo de este proyecto. Cabe anotar que estas clases, así como todos los métodos de las otras clases descritas en este capítulo se encuentran totalmente definidas en la Especificación del Perfil MIDP Ver. 2.0. En esta documentación se incluyeron nada más, las clases y métodos que fueron más relevantes para el desarrollo de esta aplicación.

CAPÍTULO III

3. SERVLETS

3.1. DESCRIPCIÓN DEL CAPÍTULO

En este capítulo se describe la teoría sobre el funcionamiento de los Servlets, así como la interacción de estos con los clientes. Esta última parte es muy importante por cuanto el sistema desarrollado se basa en el intercambio de información entre el MIDlet (cliente) y el Servlet (servidor).

Para esto en las siguientes secciones se realizará una descripción de la Arquitectura de los Servlets, la manera en que estos interactúan con los clientes, su ciclo de vida, la comunicación del servlet para utilizar y compartir otros recursos y algunos otros temas sobre esta tecnología.

3.2. INTRODUCCIÓN A LOS SERVLETS

Los servlets son módulos de código Java que se ejecutan en aplicaciones que están en el lado servidor (el nombre “Servlets” es análogo al de “Applets”, pero estos últimos se ejecutan en el lado del cliente), y tienen como objetivo atender las peticiones de los clientes. Los Servlets no están atados a un protocolo cliente-servidor específico, pero son muy utilizados con HTTP y la palabra “Servlets” es a menudo utilizada como el significado de “HTTP Servlet”.

Los servlets pueden operar en servidores que trabajan sobre diferentes plataformas, debido a que el API utilizado para implementarlos no asume nada sobre el entorno o protocolo del servidor.

Dentro de las aplicaciones que tienen los servlets, a continuación se listan las más relevantes:

- Manejo de peticiones HTTP. Por ejemplo, un servlet podría procesar los datos enviados por un cliente a través de una conexión HTTP, para luego enviar esa información a un servidor de base de datos con objeto de realizar el pago de alguna transacción. La Figura 3.1 muestra un diagrama esquemático de esta aplicación.
- Hacer factible la colaboración entre usuarios. Un servlet puede manejar múltiples peticiones concurrentes y también sincronizarlas. Esto permite a los servlets soportar aplicaciones como conferencias en línea.
- Reenviar peticiones. Los servlets pueden reenviar peticiones a otros servidores y servlets, lo que resulta muy útil a la hora de trabajar con bases de datos que se encuentran en otros servidores.

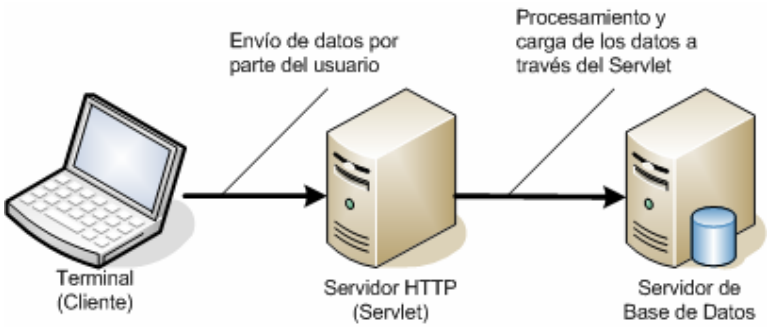

Figura. 3.1. Diagrama esquemático sobre el manejo de conexiones HTTP a través de los Servlets

3.3. ARQUITECTURA DEL PAQUETE SERVLET

El paquete **javax.servlet** proporciona todas las clases e interfases necesarias para la implementación de los servlets. A continuación se realiza una descripción de este paquete.

3.3.1. El Interfase Servlet

Todos los servlets implementan la interfase **Servlet**, bien sea directamente o extendiendo (derivando) una clase que lo implemente, como **HttpServlet**. En la Figura 3.2 se presenta un diagrama que describe lo antes mencionado.

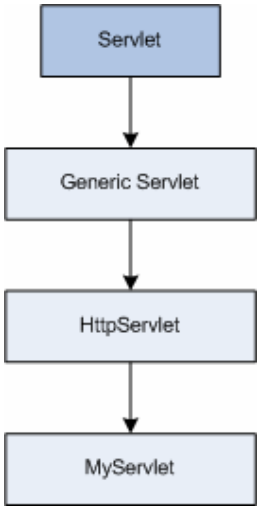

Figura. 3.2. Diagrama de flujo sobre la implementación de la interfase Servlet a través de HttpServlet

El interfase **Servlet** declara, pero no implementa ninguno de los métodos que manejan el servlet y su comunicación con los clientes.

3.3.2. Interacción con el cliente

Al momento de haberse establecido la conexión entre el cliente y el servlet, este último recibe dos objetos, que son:

- Un **ServletRequest** que encapsula la comunicación entre el cliente y el servidor.
- Un **ServletResponse** que encapsula la comunicación de respuesta, la misma que va desde el servlet hacia el cliente.

Estos dos objetos se encuentran definidos dentro del paquete **javax.servlet**.

3.3.2.1. El Interfase ServletRequest

Define un objeto que le permite al servlet recuperar la información enviada por el usuario en la petición. El contenedor del servlet crea un objeto **ServletRequest** y lo pasa como argumento al método de servicio del servlet.

Un objeto de este tipo provee de cierto tipo de información como los nombres y valores de los parámetros, atributos y un stream de entrada. Los servlets utilizan este stream para recuperar los datos enviados por los clientes a través de conexiones de tipo POST y PUT. Las interfases que se derivan (extienden) del **ServletRequest** pueden proporcionar datos adicionales sobre un protocolo específico, como por ejemplo los datos HTTP son recuperados a través de un objeto de la interfase **HttpServletRequest**.

A continuación se presentan los métodos de esta clase que fueron más relevantes para el desarrollo de este proyecto:

Método	Descripción
java.lang.Object getAttribute(java.lang.String name)	Retorna el valor del atributo especificado o nulo si no existe un atributo con ese nombre.
java.lang.String getCharacterEncoding()	Retorna el nombre de la codificación de caracteres utilizado en el cuerpo de esta petición.
int getContentLength()	Retorna la longitud, en bytes, del cuerpo de la petición o -1 si la longitud no es conocida.
java.lang.String getContentType()	Retorna el tipo MIME (Multipurpose Internet Mail Extensions) del cuerpo de la petición o nulo si esta información no es conocida.
ServletInputStream getInputStream()	Retorna el cuerpo de la petición como datos binarios utilizando un **ServletInputStream**.
java.util.Enumeration getAttributeNames()	Retorna un objeto de tipo **Enumeration** conteniendo los nombres de los atributos disponibles de esta petición.

Tabla. 3.1. Métodos de la interfase ServletRequest más relevantes para este proyecto

3.3.2.2. El Interfase ServletResponse

Define un objeto que se encarga de asistir al servlet al momento de enviar la información de respuesta al cliente. El contenedor del servlet crea un objeto **ServletResponse** y lo pasa como argumento al método de servicio del servlet.

Para enviar datos en el cuerpo de una respuesta MIME (Multipurpose Internet Mail Extensions), se utiliza un objeto de la clase **ServletOutputStream** que es retornado por el

método **getOutputStream()**. Para enviar datos en forma de caracteres se utiliza un objeto de tipo **PrintWriter** retornado por el método **getWriter()**. Para mezclar datos binarios y de texto se utilizaría un objeto **ServletOutputStream**.

A continuación se presentan los métodos más relevantes de esta clase para el desarrollo de este proyecto.

Método	Descripción
java.lang.String getCharacterEncoding()	Retorna el nombre de la codificación de caracteres utilizado para enviar la respuesta.
java.lang.String getContentType()	Retorna el tipo de contenido utilizado por el cuerpo MIME para enviar la respuesta.
ServletOutputStream getOutputStream()	Retorna un **ServletOutputStream** que permite enviar datos hacia el cliente.
java.io.PrintWriter getWriter()	Retorna un objeto **PrintWriter** que permite enviar texto hacia el cliente.
void setContentType(java.lang.String type)	Establece el tipo de contenido de la repuesta que será enviada al cliente, si es que esta no ha sido definida antes.

Tabla. 3.2. Métodos de la interfase ServletResponse más relevantes para este proyecto

3.4. INTERACCIONES CON LOS CLIENTES

Un Servlet HTTP maneja las peticiones del cliente a través del método **service(...)**. Este método soporta peticiones de cliente tipo HTTP, despachando cada petición hacia un método designado para manejarla.

3.4.1. Peticiones y Respuestas

En esta sección se describe la utilización de los objetos que representan las peticiones de los clientes y las respuestas del servidor hacia ellos, a través de los objetos de tipo **HttpServletRequest** y **HttpServletResponse** respectivamente.

3.4.1.1. Objetos HttpServletRequest

Ests objetos proporcionan acceso a los datos de cabecera HTTP, así como permite identificar el método con el que se ha realizado la petición (GET, POST, etc.). El objeto **HttpServletRequest** también permite obtener los argumentos que el cliente envía como parte de la petición.

El objeto **HttpServletRequest** realiza todas estas funciones a través de los siguientes métodos:

Método	Descripción
java.lang.String getParameter(java.lang.String name)	Retorna el valor del parámetro especificado o nulo si este no existe.
java.lang.String[] getParameterValues(java.lang.String name)	Retorna un arreglo de objetos de tipo String que contienen todos los valores que el parámetro especificado tiene.
java.util.Enumeration getParameterNames()	Retorna los nombres de los parámetros.
java.lang.String getQueryString()	Para peticiones GET de HTTP, este método devuelve en un String una línea de datos desde el cliente. Se debe realizar un análisis de estos para obtener los parámetros y valores.
java.io.BufferedReader getReader()	Si se espera los datos en forma de texto, este método devuelve un **BufferedReader** utilizado para leer la línea de datos. Esto es utilizado en peticiones POST, PUT o DELETE.
ServletInputStream getInputStream()	Si se espera datos binarios, este método devuelve un **ServletInputStream** utilizado para leer la línea de datos. Esto es utilizado en peticiones POST, PUT o DELETE.

Tabla. 3.3. Métodos de HttpServletRequest más relevantes para esta sección

3.4.1.2. Objetos HttpServletResponse

Estos objetos proporcionan dos formas de enviar datos al usuario en base a los siguientes métodos:

Método	Descripción
java.io.PrintWriter getWriter()	Retorna un objeto de tipo **PrintWriter** que permite enviar texto hacia el cliente.
ServletOutputStream getOutputStream()	Retorna un objeto **ServletOutputStream** que permite escribir datos binarios en la respuesta.

Tabla. 3.4. Métodos de HttpServletResponse más relevantes para esta sección

3.4.2. Manejo de Peticiones GET y POST

Para manejar peticiones HTTP en un servlet, se extiende (deriva) la clase **HttpServlet** y se sobrescriben los métodos del servlet que manejan las peticiones HTTP que necesitamos que sean soportadas. En esta sección se describe el manejo de peticiones GET y POST.

3.4.2.1. Manejo de Peticiones GET

Este tipo de peticiones son manejados por el método **doGet(...)**. A continuación se presenta un extracto del código utilizado para manejar este tipo de peticiones:

```
public class BetProjectServlet extends HttpServlet{

  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{

        response.setContentType("text/plain");
        PrintWriter out = null;

                out = response.getWriter();
                out.print(String_de_respuesta);

            out.close();
    }
}
```

El servlet extiende la clase **HttpServlet** y se sobrescribe el método **doGet(...)**. Para responder al cliente, este método utiliza un objeto de tipo **Writer** (**PrintWriter**), el mismo que permite devolver al cliente los datos en formato texto. Previo al envío de la información, se establece el tipo de contenido que será transmitido. Luego de haber enviado los datos, el **Writer** se cierra.

3.4.2.2. Manejo de Peticiones POST

Este tipo de peticiones son manejados por el método **doPost(...)**. A continuación se presenta un extracto del código utilizado para manejar este tipo de peticiones:

```
public class BetProjectServlet extends HttpServlet{

  public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{

        response.setContentType("text/plain");
        PrintWriter out = null;

                    out = response.getWriter();
                    out.print(String_de_respuesta);

                  out.close();
    }
}
```

El servlet extiende la clase **HttpServlet** y sobrescribe el método **doPost(...)**. Al igual que en el caso anterior, este método utiliza un **Writer** del objeto **HttpServletResponse** para devolver los datos en formato texto al cliente.

3.4.3. Problemas con los Threads[4]

Los servlets HTTP normalmente pueden atender múltiples peticiones a la vez. Si los métodos de nuestro servlet trabajan con clientes que acceden a recursos compartidos, se puede administrar la concurrencia creando un servlet que maneje sólo una petición a la vez.

Para hacer que el servlet maneje solamente la petición de un cliente a la vez, se tiene que implementar la interfase **SingleThreadModel**, además de extender la clase **HttpServlet**.

Implementar esta interfase no implica escribir ningún método extra. Una vez declarado que el servlet implementa esta interfase, el servidor se asegura de que el mismo ejecute solamente un método **service(...)** a la vez.

[4] Threads: Es la unidad básica de ejecución de un programa. Un procesopuede tener varios “Threads” ejecutándose a la vez, desmpeñando un trabajo diferente. Cuando un Thread finaliza su trabajo, es suspendido y destruido.

3.4.4. Descripción de Servlets

La "Descripción del Servlet" es un String que puede describir el propósito del servlet, su autor, número de versión o la información que el desarrollador del mismo considere importante.

El método que devuelve esta información es **getServletInfo()** y por defecto retorna el valor nulo si es que el mismo no ha sido implementado; de manera que, este método debe ser sobrescrito con la información deseada.

```
public class BetProjectServlet extends HttpServlet {
...
        public String getServletInfo() {
        return "BetProjectServlet es un servlet que se encarga de retornar la " +
                "información solicitada por el móvil";
        }
        ...
}
```

3.5. ESTRUCTURA BÁSICA DE LOS SERVLETS

A continuación se presenta la estructura que comparten todos los Servlets en lo referente al código.

```
//Importación de los paquetes necesarios
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;


public class BetProjectServlet extends HttpServlet{
    //Declaración e inicialización de variables miembro

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{

          //Dentro de este método se realiza el manejo de las peticiones de tipo GET

    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{

          //Dentro de este método se realiza el manejo de las peticiones de tipo POST

    }
}
```

3.6. EL CICLO DE VIDA DE UN SERVLET

Todos los servlets tienen el mismo ciclo de vida y es el siguiente:

1. Un servidor carga e inicializa el servlet.
2. El servlet maneja cero o más peticiones del cliente.
3. El servlet se elimina. En algunos de los casos, los servidores cumplen este paso cuando se desconectan.

En el siguiente diagrama (Figura 3.3) se presenta el ciclo el de vida de los servlets.

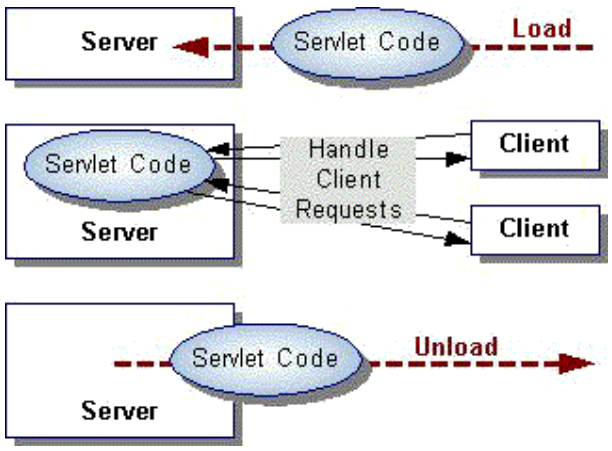

Figura. 3.3. Ciclo de Vida de los Servlets

3.6.1. Inicialización del Servlet

Cuando un servidor carga un servlet, ejecuta el método **init()** del servlet. La inicialización se completa antes de manejar peticiones de clientes y antes de que el servlet sea destruido.

Aunque muchos servlets se ejecutan en servidores multi-thread, los servlets no tienen problemas para manejar múltiples peticiones durante su inicialización. El servidor llama una sola vez al método **init()**, cuando carga el servlet y no lo llamará de nuevo a menos que vuelva a recargar el servlet. Cabe indicar que el servidor no puede recargar un servlet sin primero haberlo destruido, llamando al método **destroy()**.

3.6.2. Interactuar con los clientes

Después de la inicialización, es en esta parte donde el servlet atiende y maneja todas las peticiones de tipo GET y POST; efectuadas por los clientes. Este tema fue tratado con mayor profundidad en la sección 3.4 de este capítulo y por esta razón no se profundiza más sobre este tema.

3.6.3. Destrucción del Servlet

Los servlets se ejecutan hasta que el servidor los destruye, como por ejemplo a petición del administrador del sistema. El servidor ejecuta el método **destroy(),** del propio servlet, para su destrucción. Cabe indicar que este método se ejecuta sólo una vez. El servidor no ejecutará de nuevo el servlet hasta haberlo cargado e inicializado nuevamente.

El método **destroy()** proporcionado por la clase **HttpServlet** destruye el servlet. Para destruir cualquier recurso específico de nuestro servlet, debemos sobrescribir el método **destroy()**; por lo que, este método debería deshacer cualquier trabajo de inicialización y cualquier estado de persistencia sincronizado con el estado de memoria actual.

Un servidor llama al método **destroy()** después de que se hayan completado todas las llamadas de servicio o haya transcurrido un número específico de segundos, lo que suceda primero. Si el servlet se encuentra manejando operaciones de larga duración, los métodos de servicio deben permanecer en ejecución cuando el servidor llame al método **destroy()**.

3.7. EJECUCIÓN DE LOS SERVLETS

En esta sección se describe la configuración y posterior ejecución de servlets utilizando el servidor web **Tomcat**. Otro servidor web que puede ser utilizado es el J2EE RI de Sun.

3.7.1. Configuración de Servlets Tomcat

La configuración del servlet se lo realiza editando los elementos de un archivo de texto denominado **Web application deployment descriptor**. Este archivo debe ser nombrado

como **web.xml** y ubicado en una localidad específica donde se instala la aplicación en Tomcat. En el caso de este proyecto esta localidad fue **\betprojservlet\WEB-INF**. La información que se especifica y el formato del archivo "Web application deployment descriptor" se encuentran definidos en la Especificación Servlet Java Ver. 2.2 y 2.3.

El servlet desarrollado en este proyecto, **BetProjectServlet**, utiliza varios de los elementos del archivo descriptor, pero los más relevantes son: **servlet** y **servlet-mapping**. Todos los elementos **servlet** deben aparecer antes que los elementos **servlet-mapping**.

3.7.1.1. El Elemento Servlet

El elemento **servlet** establece un mapeo o vínculo entre el nombre del servlet y el nombre completo de la clase del servlet, como se muestra a continuación:

```
<servlet>
    <servlet-name>bet</servlet-name>
    <servlet-class>BetProjectServlet</servlet-class>
  </servlet>
```

3.7.1.2. El Elemento Servlet-Mapping

Cuando una petición es recibida por Tomcat, este debe determinar cuál servlet debe atenderla. Para esto se establecen ciertos paths o caminos llamados **alias**, que realizan la función de mapear la petición hacia un servlet específico utilizando para ello el elemento **servlet-mapping**. Los path alias aparecen después del **context root** en la URL de una petición HTTP.

El **context root** es un path (camino) que mapea hacia el **document root** de la aplicación servlet. En este caso el "context root" del servlet desarrollado es /**betprojservlet**, entonces una petición URL como: http://localhost:8084/betprojservlet/betproject; enviará la petición hacia el servlet llamado "betproject" dentro del contexto de "betprojservlet". Al momento de configurar el servidor Tomcat se debe establecer el "context root" y el "document root" para la aplicación.

A continuación se presenta la definición de este elemento para el servlet desarrollado en este proyecto:

```
<servlet-mapping>
    <servlet-name>bet</servlet-name>
    <url-pattern>/betproject</url-pattern>
  </servlet-mapping>
```

3.7.2. Configuración y manejo del Servidor Web Tomcat

Todo el proceso de configuración y manejo del Servidor Tomcat se encuentra en detalle en los archivos de soporte que vienen incluidos en el paquete de este Servidor. Dado lo extenso que podría ser esta sección, la misma no es incluida y en caso de ser un punto necesario, el lector podría referirse a los archivos antes mencionados. Por otro lado, el IDE (Integrated Development Environment) de Sun (Sun Java Studio Mobility 6) utilizado para el desarrollo de este proyecto incorpora ya, un Servidor Tomcat Ver. 5. Por las razones expuestas no se incluye más información en esta sección.

3.8. INVOCACIÓN O LLAMADO DE SERVLETS

A continuación se describen algunas formas de llamar a los servlets:

3.8.1. Ingresando la URL del servlet en una Navegador Web

Los servlets pueden ser invocados desde un Navegador Web, sin más que ingresar su URL. La URL de un servlet tiene la siguiente forma general:

```
http://server_name:port/servlet/servlet_name
```

En el caso de invocar al servlet de este proyecto, la URL sería la siguiente:

```
http://localhost:8084/betprojservlet/betproject
```

La URL de los servlets puede contener preguntas, como las peticiones GET de HTTP, de esa manera se puede acceder a secciones específicas del mismo. A continuación se

presenta un ejemplo en el cual se accede al servlet de este proyecto y a la sección donde se manejan los encuentros disponibles para realizar los pronósticos.

```
http://localhost:8084/betprojservlet/betproject?queryStr=TabApuestas
```

En donde:

`betprojservlet`: Es la carpeta donde se almacena el Servlet dentro del servidor web.
`betproject`: Representa el mapeo o vínculo que se ha definido entre el nombre del servlet y el nombre completo de la clase del servlet (`BetProjectServlet.class`).
`queryStr`: Es una variable y dependiendo del valor que tome, la petición del cliente es direccionada hacia un bloque de sentencias dentro del Servlet.

3.8.2. Invocando a un servlet desde dentro de una página HTML

Las URLs de los servlets pueden ser utilizadas en etiquetas HTML, donde se podría encontrar la URL de un script CGI-bin o la de un fichero. Para invocar a un servlet desde el interior de una página HTML, se utiliza la URL de este, en la etiqueta HTML apropiada.

3.8.3. Invocando a un servlet desde otro servlet

Los servlets pueden llamar a otros servlets. Si los dos se encuentran en distintos servidores, el uno puede realizar peticiones de tipo HTTP al otro. Por el contrario, si los dos servlets se están ejecutando en el mismo servidor, entonces uno de ellos puede acceder a los métodos públicos del otro, de manera directa.

Hasta aquí se ha presentado y descrito la teoría de funcionamiento de los Servlets. El código del servlet desarrollado para este proyecto, **BetProjectServlet.java**, así como el resto de clases y archivos implementados se incluyen como anexos.

CAPÍTULO IV

4. DESARROLLO DEL SISTEMA

4.1. DESCRIPCIÓN DEL CAPÍTULO

En este capítulo se presentan los requerimientos que debe cumplir la aplicación, así como las funcionalidades que deben ser incluidas en la misma, de manera que la versión final sea fácil de manejar para los usuarios, ya sea que tengan o no experiencia en el manejo de teléfonos móviles, así como agradable.

Luego de analizar y evaluar todas estas características, se procede a presentar una alternativa de diseño, la misma que trata de reunir todas las funcionalidades del caso para poder cumplir con los requerimientos propuestos. Posteriormente se describe en detalle todas las fases del desarrollo del sistema y en ciertos casos se incluye un extracto del código fuente para realizar una explicación más detallada.

La aplicación que utiliza el móvil será desarrollada utilizando la Plataforma J2ME de Sun Microsystems, basada en la CLDC-1.1 y con las APIs de MIDP-2.0. La aplicación que será hospedada en el Servidor es un Servlet, mientras que en el Servidor en sí se utilizó el contenedor Apache Jakarta Tomcat Ver. 5.0.19, el mismo que implementa las especificaciones Servlet 2.4 y JavaServer Pages 2.0.

4.2. DISEÑO DEL SISTEMA

La aplicación desarrollada debe cumplir con los requerimientos planteados al inicio del proyecto y que son los siguientes:

- Registro de usuario.

- Visualización, en tiempo real, de un extracto de la tabla de resultados.
- Realización de la apuesta.

Para que la aplicación cumpliese estos objetivos se pensó en la implementación de un menú principal, en el cual estuvieran las diferentes opciones para que el usuario pudiera realizar su registro, visualizar el extracto de los resultados y realizar la apuesta.

En este menú también se incluyeron algunas funcionalidades, de manera que la aplicación sea un poco más amigable para el usuario. Estas funcionalidades se listan a continuación:

- Visualización del pronóstico efectuado por el usuario.
- Ayuda sobre la utilización de la aplicación.

El esquema de apuestas utilizado para el desarrollo de este sistema fue basado en la conocida y popular lotería del fútbol "Súper Golazo". Por lo tanto, las opciones del menú, algunas de las reglas aplicadas al pronóstico, datos de la tabla de posiciones, etc.; fueron extraídos de dicho juego de apuestas.

A continuación en la Figura 4.1 se presenta el menú de opciones propuesto:

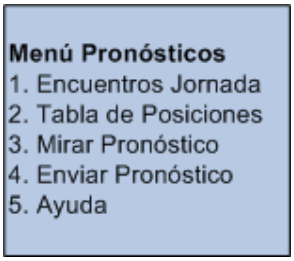

Figura. 4.1. Menú de opciones propuesto para la aplicación

La función que desempeña cada una de las opciones de este menú serán las siguientes:

1. **Encuentros Jornada:** En esta opción se presentan los encuentros de la jornada sobre los cuales el usuario puede realizar su apuesta o pronóstico.

2. **Tabla de Posiciones:** Aquí se presenta un extracto de la tabla de resultados anteriores, sobre los cuales el usuario puede basar su criterio y realizar la apuesta.
3. **Mirar Pronóstico:** Esta opción permite visualizar el pronóstico que ha realizado el usuario sobre las diferentes alternativas presentadas, de modo que pueda realizar algún cambio en caso de ser necesario.
4. **Enviar pronóstico:** En esta opción el usuario puede registrar sus datos de identificación (user y password), de manera que el sistema pueda validarlos y permita continuar con la transacción de pago.
5. **Ayuda:** Presenta una descripción de las diferentes opciones listadas más arriba y que se encuentran contenidas en el menú, así como algunas características del sistema de apuestas.

De esta manera, las opciones 1 y 4 de este menú cumplen con el primer y tercer objetivo, mientras que la opción 2 permite cumplir con el segundo. Las opciones 3 y 5 sirven de referencia y ayuda para el usuario.

Esta es la alternativa de diseño que se plantea desarrollar para el equipo celular. Por otro lado, la aplicación que será utilizada en el servidor debe gestionar todos los servicios del sistema, como son: envío de datos al móvil, validación y almacenamiento de la información enviada por el usuario (pronóstico). Para esto se optó por desarrollar un Servlet que trabajara conjuntamente con un servidor de base de datos, en el que se almacenará toda la información de este sistema, tal como: encuentros de la jornada, tabla de posiciones, datos de usuario y pronósticos. El servidor de base de datos que se utilizará es MySQL Ver. 4.1.11.

4.3. DESARROLLO DEL MIDLET Y EL SERVLET

Luego de haber analizado las características y funcionalidades del sistema, así como su solución de diseño, a continuación se presenta un diagrama (Fig. 4.2) esquemático del sistema en su totalidad:

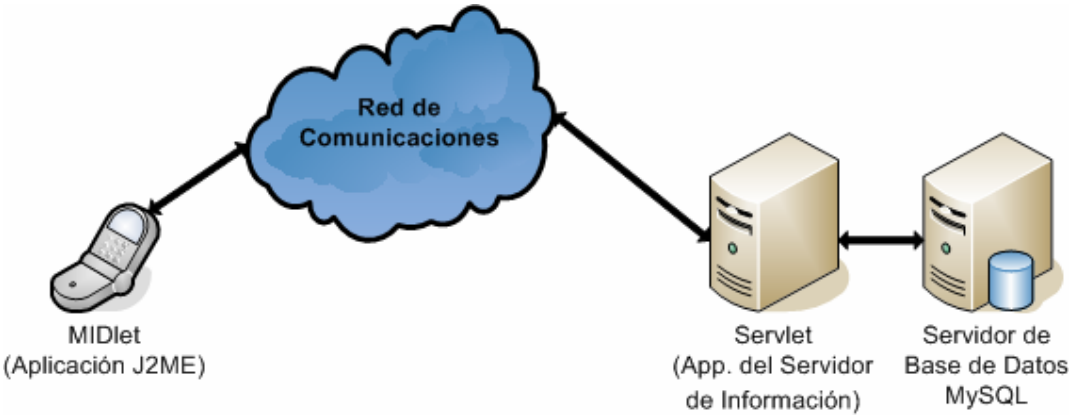

Figura. 4.2. Diagrama esquemático del sistema desarrollado

El sistema desarrollado está basado en una arquitectura cliente-servidor. La aplicación del teléfono móvil, MIDlet, realiza las consultas necesarias para recopilar la información de los pronósticos a un servidor de información, Servlet. Este último a su vez consulta la información solicitada por el móvil en una base de datos, la misma que se encuentra en un servidor de base de datos MySQL.

A continuación se realiza una descripción sobre las dos aplicaciones desarrolladas, el MIDlet y el Servlet, que conforman la totalidad el sistema:

4.4. MIDLET

Para poder implementar el menú de opciones propuesto para el MIDlet, el desarrollo del mismo está basado en un modelo de nodos o de árbol (tree model) con el fin de poder gestionar la navegación sobre las diferentes pantallas de la aplicación y que conforman las opciones del menú. Este modelo de nodos consiste en la definición de un nodo raíz o padre y sus respectivos nodos descendientes y estos a su vez pueden tener otros nodos (descendientes), por último toda esta estructura está ligada de manera jerárquica. Los nodos y nodos descendientes representan una pantalla de la aplicación, aunque puede darse el caso de que un nodo maneje más de una pantalla, pero si fuera así dicha pantalla sería una notificación o aviso (**Alert)** y no otro nodo. En la Figura 4.3 se presenta un diagrama sobre el modelo de nodos descrito anteriormente.

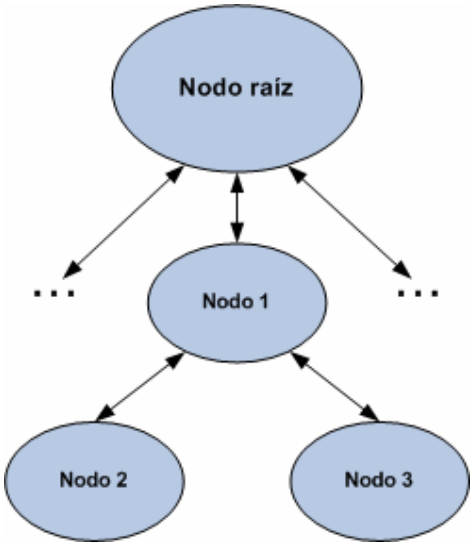

Figura. 4.3. Esquema del modelo de nodos o árbol

Para poder implementar este modelo de programación, se implementaron las clases: **SubForm** y **SubList**.

- **SubForm.** Esta clase fue derivada de la superclase **Form**. Permite definir nuevas clases que hacen factible que sus objetos derivados puedan ser vinculados entre si, de manera jerárquica. Para esto **SubForm**, incorpora una variable y un método miembro que permiten definir cuál es el nodo del cual ha sido derivado (raíz o padre) y el mecanismo para regresar a él. Cada vez que se necesite crear un nodo, se puede definir un objeto derivado de esta clase, el mismo que representa una pantalla de la aplicación.

A continuación se presenta la implementación de la clase **SubForm**.

Código de la clase SubForm.java:

```
package BetProjectMIDlet;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class SubForm extends Form implements CommandListener{

    Command backCmd;         //Botón para regresar a la pantalla/nodo anterior
```

```
    BetProjectMIDlet midlet; //El MIDlet
    Displayable parent;      //La pantalla/nodo anterior

    //Constructor
    public SubForm(String title, BetProjectMIDlet midlet, Displayable parent){
        super(title);
        this.midlet = midlet;
        this.parent = parent;

        backCmd = new Command("Atrás", Command.BACK, 1);

        addCommand(backCmd);
        setCommandListener(this);
    }

    /**
     * Called when user action should be handled
     */
    public void commandAction(Command command, Displayable displayable) {
        if(command == backCmd){
            if(parent != null)
                midlet.myDisplay.setCurrent(parent);     //Presenta el nodo anterior
        }
    }
}
```

El campo `Displayable parent` permite guardar una referencia del nodo anterior (padre), de manera que se pueda regresar al mismo luego de haber finalizado cualquier acción o evento en el nodo actual, mientras que el campo `BetProjectMIDlet midlet` es una referencia al MIDlet, lo cual permite acceder a cualquiera de sus campos o métodos miembro y en especial al campo `myDisplay` que es el encargado de presentar el objeto `Displayable` necesario en pantalla.

- **SubList.** Esta clase es similar a **SubForm**, pero la diferencia está en que **SubList** fue derivada de la superclase **List**, por cuanto fue pensada para otras aplicaciones en donde se necesitaban las funcionalidades que nos proporciona esta superclase. El funcionamiento y el propósito que tiene **SubList** es el mismo que **SubForm**.

Cabe indicar que se puede definir objetos de tipo **SubList** o **SubForm** con el fin de crear nuevos nodos (pantallas), la consideración que se debe realizar para ello es el uso que estos nodos van a tener en la aplicación.

A continuación se presenta un diagrama esquemático del menú de opciones implementado utilizando este modelo de nodos.

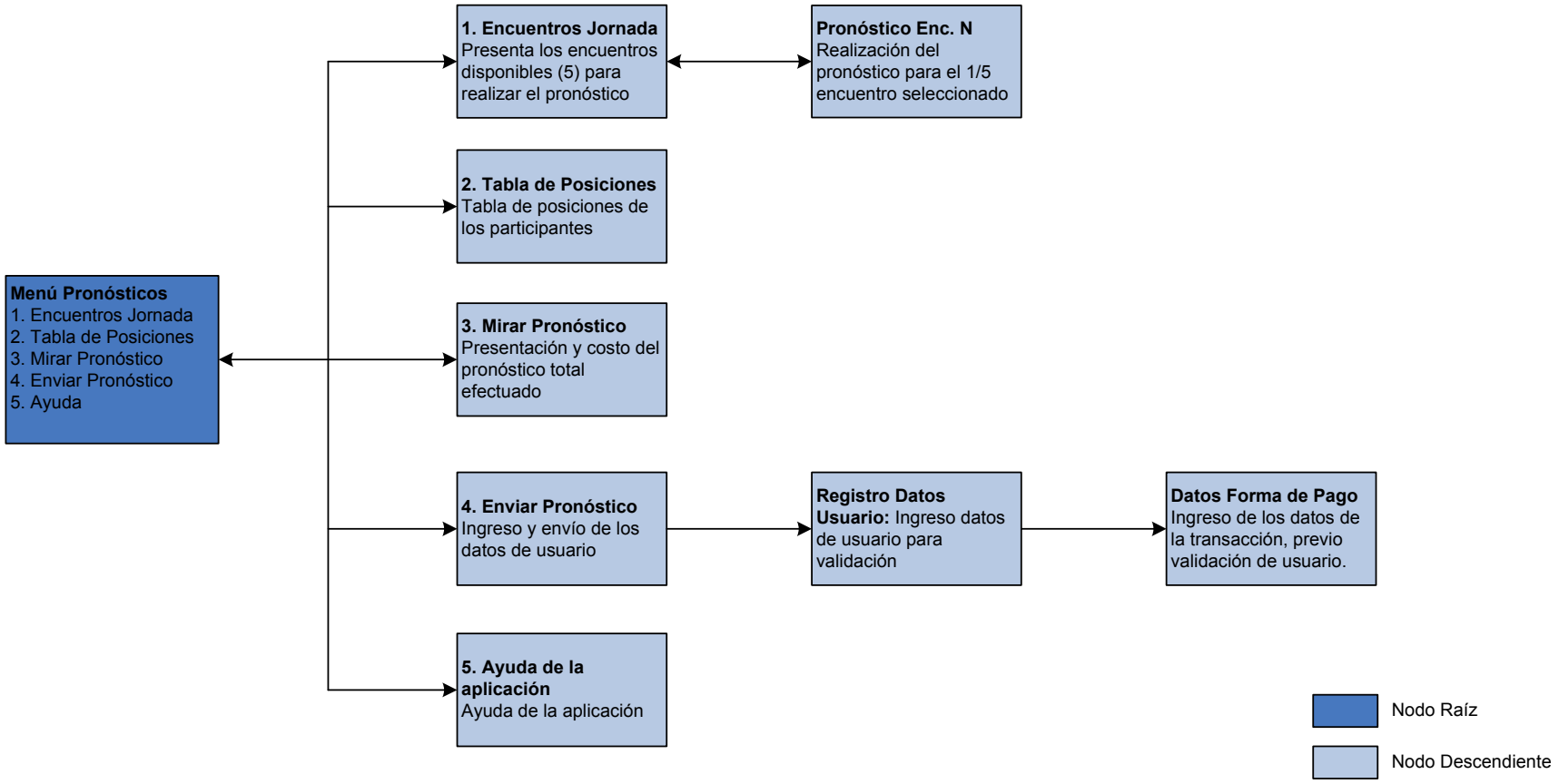

Figura. 4.4. Esquema del Menú de Opciones empleando el Modelo de Nodos

Como se puede apreciar en la Figura 4.4, el "Menú Pronósticos" representa junto con sus respectivas opciones al **Nodo Raíz**, desde donde se extienden los nodos descendientes y de algunos de ellos se extienden otros nodos (descendientes).

La primera pantalla que presenta el MIDlet es el "Menú Pronósticos"; es decir el "Nodo Raíz" de la aplicación y desde donde parten las demás opciones del menú, representando cada una de ellas un nodo. Esta pantalla fue implementada utilizando un objeto de tipo **List** en el cual se incluyeron todas las opciones del menú principal. El código del MIDlet, **BetProjectMIDlet.java**, se incluye como anexo. En la Figura 4.5 se presenta la pantalla principal de la aplicación:

Figura. 4.5. Menú Pronósticos (Pantalla principal)

A continuación se realizará una descripción acerca de la implementación de cada una de las opciones del menú:

Opción 1. Encuentros Jornada. Esta opción permite visualizar los encuentros disponibles sobre los cuales se puede realizar el pronóstico. La información sobre los encuentros reside en un servidor de base de datos MySQL y el acceso a ella se lo hace a través del Servlet. Para esto el móvil realiza una conexión de tipo HTTP hacia el Servidor de Información que es dónde se encuentra el Servlet y este a su vez realiza una conexión y posterior consulta a la base de datos para luego retornar la información al móvil. En la Figura 4.6 se presenta un impreso de la pantalla de esta opción.

Figura. 4.6. Pantalla del Menú Encuentros Jornada

Para esta opción se implementó la clase **TbEncuentros.java**, derivada de la clase **SubList**. Esta clase se encarga de gestionar la conexión HTTP, luego de ello presenta la información devuelta por el servidor en pantalla o en caso contrario, si es que se ha presentado algún error en dicha conexión no presenta nada. También se podría dar el caso de que el servidor tenga algún inconveniente y no permite acceder a su contenido, como resultado de esto la aplicación igualmente no presentaría ninguna información en pantalla.

Luego de presentar la información, el usuario puede seleccionar cualquiera de los encuentros para realizar el pronóstico. Una vez seleccionado alguno de ellos, se presenta otra pantalla donde se puede realizar el pronóstico del encuentro, esto se lo hace marcando las opciones que el usuario desee, como se lo puede apreciar en la Figura 4.7.

Para esta acción se implementó la clase **TbApuestas**, igualmente derivada de la clase **SubList** y que se encuentra dentro de la clase **TbEncuentros.java**. Esta clase divide al encuentro seleccionado en sus respectivas opciones para realizar el pronóstico, esto se lo hace marcando el casillero de la opción deseada. Luego de marcar las opciones se debe presionar el botón “Guardar”, esto hace que los datos se almacenen en la memoria persistente del teléfono para en lo posterior permitir editarla y en el proceso final enviar esta información al servidor. El almacenamiento de los datos en la memoria persistente del teléfono se lo hace a través del **RMS (Record Management System)**.

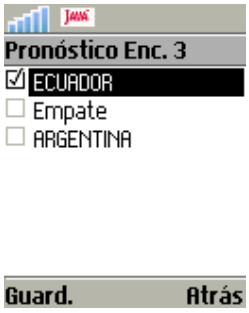

Figura. 4.7. Realización del pronóstico para el encuentro 3

Para administrar la información almacenada en la memoria del teléfono a través del RMS se implementó la clase **RmsDataManager.java**, la misma que a través de un objeto **RecordStore** permite agregar, editar y realizar otras distintas operaciones con los datos de los pronósticos.

El tema de la comunicación entre el móvil y el servidor, así como el almacenamiento de la información y gestión de la misma utilizando el RMS, se los tratará con mayor profundidad en los puntos 4.4.1 y 4.4.2.

Opción 2. Tabla de Posiciones. Esta opción permite al usuario visualizar la tabla de posiciones de los participantes, así como información adicional sobre el puntaje que poseen. Toda esta información al igual que la anterior reside en un servidor de base de datos MySQL y el acceso a ella desde el MIDlet se lo hace a través del Servlet. En la Figura 4.8 se presenta un impreso de la pantalla de esta opción.

Para esta opción se implementó la clase **TbPositions.java**, derivada de la clase **SubForm**. Esta clase se encarga de gestionar la conexión HTTP, luego de ello presenta la información devuelta por el servidor en pantalla, caso contrario si es que se ha presentado algún error en dicha conexión no presenta nada.

Figura. 4.8. Pantalla de la opción Tabla de Posiciones

Opción 3. Mirar Pronóstico. Presenta información sobre el pronóstico efectuado y el costo total del mismo. Esta información es extraída de la memoria persistente del teléfono a través del objeto **RmsDataManager** creado para administrar los datos a través del RMS. En la Figura 4.9 se presenta un impreso de la pantalla de esta opción, luego de haber realizado un pronóstico sobre todos los encuentros disponibles.

Figura. 4.9. Pantalla de la opción Mirar Pronóstico

En este caso se implementó la clase **ForecastViewer.java**, la misma que se encarga de consultar la información almacenada en la memoria persistente del móvil y calcular el costo total del pronóstico, para luego presentarla al usuario. **ForecastViewer.java** es una clase que se deriva de la clase **SubForm**.

Opción 4. Enviar Pronóstico. Aquí es donde el usuario, luego de haber realizado todos los pronósticos sobre los encuentros deseados, puede ingresar y enviar sus datos de identificación hacia el servidor, donde podrán ser validados. En caso de que esta acción sea exitosa, se podrá pasar a seleccionar e ingresar los datos de la forma de pago de la transacción, luego de lo cual esta información y la de los pronósticos serán enviadas al

servidor (Servlet). Cabe indicar que luego de cada intento erróneo de acceder al sistema se presentará una notificación de error y en la tercera y última, el sistema borrará toda la información almacenada (pronósticos) y regresará a la etapa inicial de su ejecución. La acción de eliminar la información almacenada luego de los 3 intentos erróneos, fue implementada por motivos de seguridad para el dueño del terminal. En la Figura 4.10 tenemos la pantalla inicial del proceso de registro del usuario, que es en donde se deben ingresar los datos de user y password.

Figura. 4.10. Pantalla inicial de la opción Enviar Pronóstico

Para la primera parte de esta opción del menú, "Registro Datos Usuario", se implementó la clase **RegDataUser.java**, derivada de la clase **SubForm**, la misma que codifica los datos de identificación ingresados por el usuario a través del esquema de codificación **Base64**[5], esto con el fin de evitar la transmisión de los datos originales a través de un canal de comunicaciones poco o nada confiable; es decir, susceptible a intromisiones por terceros o agentes no deseados. Luego de lo cual establece una conexión HTTP con el servidor para poder transmitir esta información.

Posteriormente se pasa a la selección e ingreso de datos correspondientes a la forma de pago ("Datos Forma de Pago"), la misma que podrá ser realizada únicamente a través de un medio electrónico de pago, como lo es una tarjeta de crédito. En la Figura 4.11 se presenta un impreso de la pantalla donde se deben ingresar estos datos.

[5] Codificación Base64: Es un algoritmo de codificación de datos comúnmente utilizado en el esquema de autenticación básico de la especificación HTTP 1.1 y se encuentra definido en el RFC (Request For Comments) 3548.

Figura. 4.11. Ingreso de datos de la forma de pago (Op. Enviar Pronóstico)

Para esto se implementó la clase **PaymentForm**, dentro de la clase **RegDataUser.java**, la misma que se encarga de presentar las opciones de la forma de pago, para luego establecer una conexión con el servidor y poder transmitir la información (pronósticos), que se encuentra almacenada en la memoria del teléfono. Posteriormente esta información será almacenada en la base de datos mediante el Servlet. Cabe indicar que en esta versión del MIDlet, la información de la forma de pago no es considerada para realizar ninguna acción, únicamente se habilitó esta opción con el fin de simular un pago electrónico.

Opción 5. Ayuda. Para lograr que la aplicación sea un poco más fácil de utilizar, se incluyó dentro del menú principal una opción de ayuda, en la que se presenta una breve explicación sobre cada una de las opciones del menú, así como algunos lineamientos y consideraciones de este juego de apuestas. A continuación se presenta un impreso de la pantalla de esta opción en la Figura 4.12.

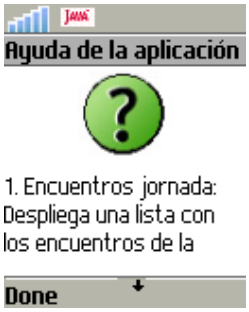

Figura. 4.12. Pantalla de la opción de Ayuda de la aplicación

Esta opción de ayuda fue implementada a través de un objeto de tipo **Alert**, el mismo que se encarga de presentar esta información hasta que el usuario presione la tecla **Done**.

Hasta este punto se ha hecho una explicación de todas las opciones disponibles en el menú principal de la aplicación, así como también se indica las clases que fueron implementadas para gestionar cada una de las opciones. El código fuente del MIDlet, así como de todas estas clases se han incluido como anexo en la documentación de este proyecto.

4.4.1. Comunicación cliente-servidor entre el MIDlet y el Servlet

Una de las ventajas que tienen los dispositivos MIDP (Capítulo 2) es la capacidad que tienen para conectarse con servidores de información, bases de datos o con la Web con el objetivo de enviar o recibir información, además de que lo pueden hacer en cualquier sitio o momento gracias a la movilidad que nos ofrecen. Cabe recalcar que estos dispositivos también tienen algunas limitaciones, como son la baja capacidad de procesamiento, almacenamiento de información y también las limitaciones que tienen para presentar cierta información en sus displays.

Para implementar aplicaciones MIDP que tengan esta capacidad de conexión se utilizan los paquetes (clases e interfases) `javax.microedition.io` y `java.io`. El primer paquete proporciona numerosas herramientas que nos permiten gestionar y administrar diferentes tipos de conexiones de red y que podrían ser de tipo: HTTP, datagramas, sockets, entre otros. El segundo paquete nos proporciona las herramientas necesarias para manejar el flujo (stream) de datos que se envían o reciben por medio de estas conexiones; es decir, nos permiten encapsular la información a ser transmitida y también recuperar la información que recibimos.

El paquete `javax.microedition.io` define el **Generic Connection Framework (GCF)** que nos es más que una colección de interfases que permite cumplir con las necesidades de conectividad de los dispositivos MIDP, ya que estos a más de tener características comunes tienen otras que son específicas y que también necesitan atención. En la Figura 4.13 se presenta la estructura jerárquica del GCF.

Como se puede apreciar en la parte superior de esta estructura se encuentra la interfaz **Connection**, la misma que representa la conexión más genérica y abstracta que se puede

implementar. El resto de interfases que se derivan de **Connection** son los diferentes tipos de conexión que se pueden implementar. Dado que los dispositivos MIDP poseen bastantes restricciones, se hace imposible implementar los distintos protocolos de comunicación, sino que se implementa únicamente el protocolo requerido. Por esto el GCF nos provee de la clase **Connector**, la misma que nos permite crear cualquier tipo de conexión, ya sea: HTTP, socket, etc. A continuación se realiza una breve descripción de la clase e interfases que fueron utilizadas en este proyecto, pertenecientes al GCF.

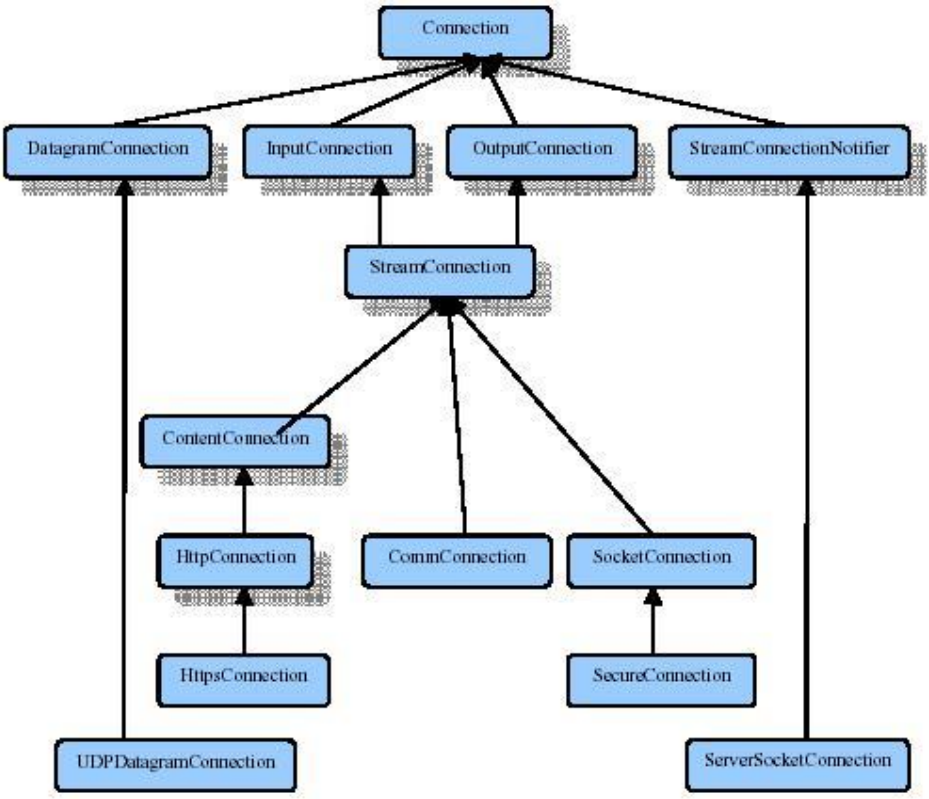

Figura. 4.13. Estructura jerárquica del Generic Connection Framework (GCF)

Clase Connector. El GCF nos provee de esta clase para realizar cualquier tipo de conexión, sin tener que preocuparnos de cómo se implemente el protocolo necesario. La cadena de parámetros que describen la dirección hacia donde se quiere realizar la

conexión, debe cumplir con el formato URL descrito en el RFC[6] 2396, el mismo que tiene la siguiente forma:

```
{scheme}:[{target}][{parms}]
```

Donde:

`{scheme}`: Es el nombre del protocolo, como por ejemplo: http.

`{target}`: Representa generalmente alguna clase de dirección de red.

`{parms}`: Forma una serie de equivalencias a través del formato ";x=y", como por ejemplo: ";type=a".

De tal manera que la conexión se realiza mediante el siguiente mensaje genérico:

```
Connector.open("protocol:address;parameters");
```

Un ejemplo de este mensaje sería:

```
Connector.open("http://www.betproject.com/betproject?Str=TabApuestas");
```

La clase **Connector** es la encargada de buscar la clase específica que implemente el protocolo requerido y si la encuentra el método **open()** devuelve un objeto que implementa la interfaz **Connection**. En la Tabla 4.1 se presentan los métodos de la clase **Connector**.

Método	Descripción
static Connection open(String name)	Crea y abre una conexión.
static Connection open(String name, int mode)	Crea y abre una conexión con permisos.
static Connection open(String name, int mode, boolean timeouts)	Crea y abre una conexión especificando el permiso y el tiempo de espera.
static DataInputStream openDataInputStream(String name)	Crea y abre una conexión de entrada devolviendo para ello un DataInputStream.

Tabla. 4.1. Métodos de la Clase Connector

[6] RFC (Request For Comments): Es un documento que especifica un estándar de Internet y busca su discusión y sugerencias para que este pueda ser mejorado.

Método	Descripción
static DataOutputStream openDataOutputStream(String name)	Crea y abre una conexión de salida a través de un DataOutputStream.
static InputStream openInputStream(String name)	Crea y abre una conexión de entrada utilizando un InputStream.
static OutputStream openOutputStream(String name)	Crea y abre una conexión de salida devolviendo para ello un OutputStream.

Métodos de la Clase Connector (*Continuación Tabla 4.1*)

En la Tabla 4.2 se presentan los permisos que se pueden fijar en la conexión, dependiendo del protocolo utilizado.

Modo	Descripción
READ	Permiso de sólo lectura.
READ_WRITE	Permiso de lectura y escritura.
READ	Permiso de sólo escritura.

Tabla. 4.2. Permisos que se pueden fijar en la conexión

4.4.1.1. Interfases del GCF

1. Interfaz Connection. Es la interfase que se encuentra en lo más alto de la jerarquía del GCF, por esta razón el resto de interfases se deriva de esta interfaz. Una conexión de tipo **Connection** se crea después de que un objeto de tipo **Connector** invoque al método **open()**. Este interfaz representa el tipo más básico de una conexión genérica y es por esta razón que implementa únicamente un método. Este método es **close()** y se encarga de cerrar la conexión.

2. Interfaz InputConnection. Esta interfase define los recursos necesarios para gestionar una conexión basada en streams (Flujo o cadena de datos) de entrada. Posee solamente dos métodos que devuelven objetos de tipo **InputStream** y que se listan en la Tabla 4.3.

Método	Descripción
DataInputStream openDataInputStream()	Abre y retorna un stream de datos de entrada para una conexión.
InputStream openInputStream()	Abre y retorna un stream de entrada para una conexión.

Tabla. 4.3. Métodos del Interfaz InputConnection

3. Interfaz OutputConnection. Define los recursos necesarios para gestionar una conexión basada en streams (Flujo o cadena de datos) de salida. Posee solamente dos métodos que devuelven objetos de tipo **OutputStream** y que se listan en la Tabla 4.4.

Método	Descripción
DataOutputStream openDataOutputStream()	Abre y retorna un stream de datos de salida para una conexión.
OutputStream openOutputStream()	Abre y retorna un stream de salida para una conexión.

Tabla. 4.4. Métodos del Interfaz OutputConnection

4. Interfaz StreamConnection. Esta interfaz hereda los métodos de las dos interfases que están por encima de ella, **InputConnection** y **OutputConnection**, por tal motivo no añade ningún método nuevo. Lo que hace es representar una conexión basada en streams de entrada y salida.

5. Interfaz ContentConnection. Representa las conexiones que pueden describir de alguna manera su contenido, contrario a los tipos de conexión anteriormente descritos en donde se transmitían bytes sin importa su estructura. En este tipo de conexión, el tipo de información a ser transmitida debe ser conocida previamente. En conclusión, esta interfaz representa a familias de protocolos en los que se definen atributos, los cuales describen los datos que se transportan. Los métodos que añade esta clase se listan en la Tabla 4.5.

Método	Descripción
String getEncoding()	Retorna el tipo de codificación del contenido que el recurso conectado esta entregando.
long getLength()	Devuelve la longitud del contenido recibido.
String getType()	Retorna el tipo de contenido que el recurso conectado esta entregando.

Tabla. 4.5. Métodos del Interfaz ContentConnection

6. Interfaz HttpConnection. En esta interfaz se basó la comunicación entre el MIDlet y el Servlet, puesto que define los métodos y constantes necesarios para una conexión HTTP. El protocolo HTTP es un protocolo de tipo petición-respuesta, donde el cliente realiza una petición y espera la respuesta del servidor. Este tipo de comunicación es la que se lleva a cabo entre un navegador web (cliente) y un servidor web (servidor), en el caso de este proyecto la comunicación se lleva a cabo entre el MIDlet y el Servlet, como se mencionó antes. Cabe indicar que los parámetros de la conexión deben ser fijados antes de que la petición sea enviada.

4.4.1.2. Estados de la conexión HTTP

La conexión HTTP se puede encontrar en alguno de los tres estados siguientes: establecimiento de conexión, conectado, conexión cerrada. En la Figura 4.14 se presenta un diagrama explicando el flujo de estos estados.

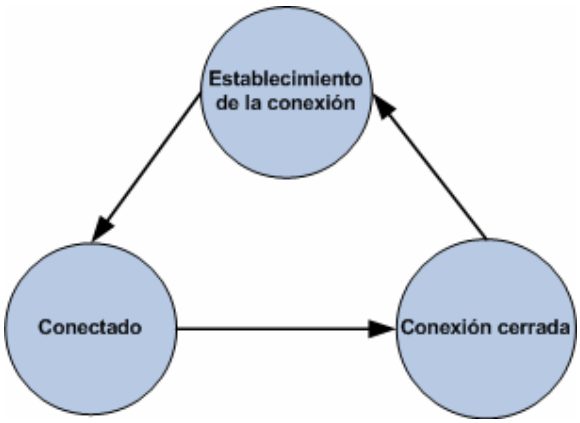

Figura. 4.14. Estados de la conexión HTTP

A continuación se realiza una descripción de cada uno de estos tres estados.

1. Establecimiento de la conexión. En este estado se fijan los parámetros para la conexión. En la Tabla 4.6 se listan los métodos que pueden ser invocados solamente en el estado de establecimiento.

Método	Descripción
void setRequestMethod(String method)	Establece el método de petición URL y puede ser: GET, POST o HEAD.
void setRequestProperty(String key, String value)	Establece las propiedades generales.

Tabla. 4.6. Métodos que pueden ser invocados en el establecimiento de la conexión

El primer método permitir establecer el tipo de conexión que se va a realizar y que puede ser de tipo: GET, POST o HEAD. El método POST es usado, en lugar del método GET, para transmitir datos de forma arbitraria desde el cliente hacia el servidor. Con el método GET, la única manera de pasar datos es utilizando texto codificado como parte de la petición URL. Por el contrario, el método POST es más flexible porque permite especificar un determinado formato de dato (inclusive binario) y no tiene la misma restricción o desventaja en cuanto a la longitud de los datos que tiene el método GET.

El segundo método permite definir cierta información adicional a la petición, la misma que permite negociar con el servidor detalles como por ejemplo: idioma, tiempos de respuesta, formato de la respuesta, etc. Estos campos forman parte de la cabecera de petición y se puede encontrar la definición de todos estos en el RFC 2068. En la Tabla 4.7 se listan algunos de estos campos que son más relevantes para este proyecto.

Campo	Descripción
Accept	Formatos que acepta el cliente.
Connection	Indica al servidor si se quiere cerrar la conexión después de la petición o se quiere dejar abierta.
Content-Language	País e idioma que utiliza el cliente.

Tabla. 4.7. Campos de la cabecera de petición URL

Campo	Descripción
Content-Type	Indica el tipo de contenido que tiene el cuerpo de la entidad enviada.
User-Agent	Tipo de contenido que devuelve el servidor.

Campos de la cabecera de petición URL (*Continuación Tabla 4.7*)

2. Estado de conexión (Conectado). La transición entre el estado de Establecimiento de la conexión y el de Conexión se produce cuando uno de los métodos necesita enviar datos hacia o recibirlos desde el servidor. A continuación se listan los métodos que producen esta transición:

- `openInputStream`
- `openDataInputStream`
- `getLength`
- `getType`
- `getEncoding`
- `getHeaderField`
- `getResponseCode`
- `getResponseMessage`
- `getHeaderFieldInt`
- `getHeaderFieldDate`
- `getExpiration`
- `getDate`
- `getLastModified`
- `getHeaderField`
- `getHeaderFieldKey`

Luego de que el servidor haya procesado la petición del cliente, tendrá que enviar devuelta una respuesta, la misma que se compone de tres secciones y que son las siguientes:

- Línea de estado
- Cabecera
- Cuerpo

En la sección de la **Línea de estado**, el servidor HTTP retorna información que indica el éxito o falla de la petición del cliente. Esta información se basa en códigos de estado, los mismos que se dividen en las siguientes categorías:

- 1xx – Información
- 2xx – Éxito
- 3xx – Redirección
- 4xx – Error en el lado del cliente
- 5xx – Error en el lado del servidor

La línea de estado completa que retorna el servidor incluye la siguiente información: versión del protocolo HTTP del servidor, código de estado y el texto del mensaje, como se muestran en los siguientes ejemplos:

- "HTTP/1.1 200 OK"
- "HTTP/1.1 400 Bad Request"
- "HTTP/1.1 500 Internal Server Error"

Los métodos que nos proporciona la interfaz **HttpConnection** para obtener esta información se presenta en la Tabla 4.8.

Método	Descripción
int getResponseCode()	Retorna el código de estado de la respuesta HTTP.
String getResponseMessage()	Devuelve el mensaje de respuesta.

Tabla. 4.8. Métodos que retornan información de la Línea de estado

En la sección de **Cabecera**, de igual manera que el cliente, el servidor puede enviar información a través de los campos de cabecera. Para poder obtener estos datos que son pasados como parte de la respuesta del servidor, el cliente puede invocar a los siguientes métodos (Tabla 4.9).

Método	Descripción
String getHeaderField(int n)	Obtiene el valor del campo de la cabecera mediante su índice.
String getHeaderField(String name)	Obtiene el valor del campo de la cabecera mediante su nombre.
String getHeaderFieldKey(int n)	Obtiene el nombre del campo de cabecera usando su índice.

Tabla. 4.9. Métodos que retornan información de la Cabecera

Por último en la sección de **Cuerpo**, el servidor puede enviar información adicional al cliente como un stream de datos.

3. Estado de Conexión cerrada. Para terminar o cerrar la conexión con el servidor, el cliente (MIDlet) debe invocar al método **close()**.

Todos los métodos y campos de la interfaz **HttpConnection** del GCF se encuentran definidos en el Perfil MIDP 2.0. Arriba se presentaron los métodos más relevantes para el desarrollo de este proyecto.

4.4.1.3. Implementación de la conexión HTTP

Luego de haber proporcionado una base de conocimientos sobre la comunicación HTTP entre un MIDlet y un Servlet, a continuación se describe como fueron implementadas las opciones de: Encuentros Jornada, Tabla de Posiciones y Enviar Pronóstico; del menú principal de la aplicación que necesitaron este tipo de conexión (HTTP).

La implementación de las dos primeras opciones del menú, “Encuentros Jornada” y “Tabla de Posiciones”, fue similar. Esto por cuanto las dos presentan al usuario información en tiempo real que es provista por el servidor a través de una conexión HTTP. Las clases que manejan estas dos opciones: **TbEncuentros.java** y **TbPositions.java;** respectivamente, realizan una conexión HTTP de tipo GET a través de un **Thread** hacia el servidor con el fin de que este les devuelva la información necesaria. Un **Thread** es un flujo secuencial de control dentro de un programa y se lo utiliza para aislar tareas. En la

Tabla 4.10 se presentan los valores de los campos de la cabecera de petición URL para estas dos conexiones.

Campo	Valor
Content-Language	es-ES
User-Agent	Profile/MIDP-2.0 Configuration/CLDC-1.0
Connection	Close

Tabla. 4.10. Campos de la cabecera de petición URL utilizados por las clases TbEncuentros.java y TbPositions.java

A continuación se presenta el código del método **run()**[7] que utiliza el **Thread** luego de haber sido iniciado, para gestionar la conexión entre el MIDlet y el Servlet:

```
public void run() {
        HttpConnection c = null;
        InputStream is = null;
        try{
            String url = midlet.getAppProperty("BaseURL") + "?queryStr=TabApuestas";
            //Estableciendo parametros para la conexion
            c = (HttpConnection)Connector.open(url);
            c.setRequestMethod(HttpConnection.GET);
            c.setRequestProperty("Content-Language", "es-ES");
            c.setRequestProperty("User-Agent", "Profile/MIDP-2.0 Configuration/CLDC-1.0");
            c.setRequestProperty("Connection", "Close");
            is = c.openInputStream(); //Transición del estado de establecimiento al de
                                      //conexión
            procesarRespuesta(c, is); //Procesamiento de la información enviada por el
                                      //servidor
        }catch(Exception e){
            System.out.println("Except: TbEncuentros/run: " + e.getMessage());
        }finally {
             try{
                 if (is != null) is.close();
                 if (c != null) c.close();
             }catch(Exception e){
                 System.out.println("Except:      Communication/run/finally/c:      "      +
e.getMessage());
             }
       }
    }
```

[7] **run()** es un método que es invocado por un objeto de tipo **thread** a través de otro método, **start()**. Se sobrescribe este método con el fin de desempeñar cualquier tipo de acción con el **thread**. El **thread** es un objeto muy utilizado en el lenguaje de programación Java.

Luego de que el método **openInputStream()** establece y devuelve un stream de entrada, la función **public void procesarRespuesta(HttpConnection c, InputStream is)** se encarga de convertir este stream de datos (bytes) en una arreglo de bytes para su posterior procesamiento y presentación.

Para la opción "Enviar Pronóstico", se implementaron las clases **RegDataUser.java** y **PaymentForm**.

RegDataUser.java: Esta clase se encarga de realizar una conexión HTTP de tipo POST a través de un "Thread", insertando en su cabecera de petición los datos de identificación del usuario para su posterior validación en el servidor. Estos datos pertenecen al campo **Authorization** y se encuentran codificados utilizando el algoritmo **Base64**, por cuanto se utilizó el Esquema Básico de Autenticación HTTP. Si la respuesta por parte del servidor es positiva, resultado de que los datos de usuario sean correctos, se recibirá el código de estado 200 (HTTP/1.1 200 OK) y se pasará a la siguiente pantalla de la aplicación donde se podrán ingresar los datos de la forma de pago. Pero si por el contrario, los datos de usuario son incorrectos se recibirá el código 401 (HTTP/1.1 401 UNAUTHORIZED) y la clase gestionará la presentación del respectivo mensaje de error, validando que no se llegué a sobrepasar los tres intentos erróneos porque de hacerlo, la aplicación eliminará toda la información almacenada y regresará a la etapa inicial. En la Tabla 4.11 se presenta los valores fijados en los campos de la cabecera de la petición URL para esta conexión.

Campo	Valor
Content-Language	es-ES
Content-Type	text/plain
Accept	text/plain
User-Agent	Profile/MIDP-2.0 Configuration/CLDC-1.0
Connection	Close
Authorization	"Basic " + datos_usuario_codificados

Tabla. 4.11. Campos de la cabecera de petición URL utilizado por RegDataUser.java

PaymentForm: Se implementó esta clase para gestionar la pantalla "Datos Forma de Pago". Una vez que los datos de la forma de pago hayan sido ingresados, esta clase

establece una conexión de tipo POST con el servidor para transmitir la información de los pronósticos efectuados por el usuario. Cabe recalcar que la información del pago no es transmitida al servidor, puesto que esta acción no fue implementada en esta versión del MIDlet.

El método **run()** del **Thread** que utiliza esta clase difiere al que utilizan las dos clases anteriores que establecen conexiones HTTP. Este método establece una petición de POST con el objetivo de enviar información, aunque luego de enviarla también recibe información por parte del servidor. A continuación se presenta el código de este método.

```
public void run() {
    HttpConnection con=null;
    OutputStream os=null;
    InputStream is=null;
    try{
        String url = midlet.getAppProperty("BaseURL") + "?queryStr=DataForescast";
        con = (HttpConnection)Connector.open(url);
        con.setRequestMethod(HttpConnection.POST);
        con.setRequestProperty("Content-Language", "es-ES");
        con.setRequestProperty("Content-Type", "text/plain");
        con.setRequestProperty("Accept", "text/plain");
        con.setRequestProperty("User-Agent", "Profile/MIDP-2.0 Configuration/CLDC-1.0");
        con.setRequestProperty("Connection", "Close");

        midlet.rsdatosEnc.resetTotalForecast();
        midlet.rsdatosEnc.readData();
        os = con.openOutputStream();
        os.write(midlet.rsdatosEnc.totalForecast.getBytes());
        os.close();
        os = null;

        is = con.openInputStream();
        procesarRespuesta(con, is);

        if(strResp.equals("1"))
            //Presenta mensaje de transacción completa
            showConfirm(true);
        else
            //Presenta mensaje de error en la transacción
            showConfirm(false);

    }catch(Exception e){
        System.out.println("Except: PaymentForm/run/try: " + e.getMessage());
    }finally {
     try{
         if (con != null) con.close();
```

```
        }catch(Exception e){
            System.out.println("Except: PaymentForm/run/finally/con: " + e.getMessage());
        }
       }
}
```

Luego de haber fijado la petición, con el método **openOutputStream()** se crea y abre un stream de salida para la conexión, el método **write(…)** se encarga de escribir los datos en este stream, para luego cerrar y liberar cualquier recurso utilizado por este a través de **close()**.

Con los métodos **openInputStream()** y **procesarRespuesta(con, is)** se valida la respuesta enviada por el servidor, acerca del éxito o fracaso de la transmisión de estos datos.

4.4.2. Record Management System (RMS)

El **Record Management System** (Sistema de Gestión de Registros) es un sistema que incorpora el MIDP para que los MIDlets puedan almacenar información de manera persistente en la memoria del teléfono, luego de lo cual se la podrá recuperar en cualquier momento. El RMS fue utilizado en este proyecto, para guardar la información de los pronósticos realizados en la opción "Encuentros Jornada" del menú principal, con el fin de evitar que se pierdan estos datos si ocurriere algún evento no previsto con el móvil, como por ejemplo si tuviera una llamada entrante. Posteriormente estos datos son recuperados para presentar la información de la opción "Mirar Pronóstico".

4.4.2.1. RecordStore

La clase **RecordStore** del paquete **javax.microedition.rms** es la que implementa este sistema de almacenamiento de datos (RMS), el mismo que está basado en registros. Un registro es un arreglo de bytes de datos. El RMS no tiene la capacidad de soportar otro tipo de dato, de manera que es el programador quien tiene la tarea de realizar la conversión de los diferentes tipos de datos a la hora de almacenar y recuperar la información.

A continuación se listan las propiedades que tienen los objetos **RecordStore (RS)**:

- El nombre de un RS es sensible a mayúsculas y minúsculas y está formado por un máximo de 32 caracteres UNICODE.
- Dentro de un **MIDlet suite**[8] no pueden existir dos **RecordStores (RSs)** con el mismo nombre.
- MIDlets del mismo suite pueden compartir RSs.
- MIDlets de diferentes suites no pueden compartir RSs.
- Cuando un MIDlet suite es removido, sus RSs también lo son.

Los registros del "RecordStore" están formados por dos partes que son las siguientes:

- Un identificador de registro o ID, que es un número entero que funciona como el índice de un vector para poder identificar a los registros. El valor inicial de este ID es uno y se incrementa de manera secuencial en una unidad cada vez que se añade un nuevo registro. Si un registro es borrado, el último ID se mantiene y el siguiente registro tendrá el ID siguiente al último, como se lo puede apreciar en la Figura 4.15.
- Un arreglo de bytes que es utilizado para almacenar los datos.

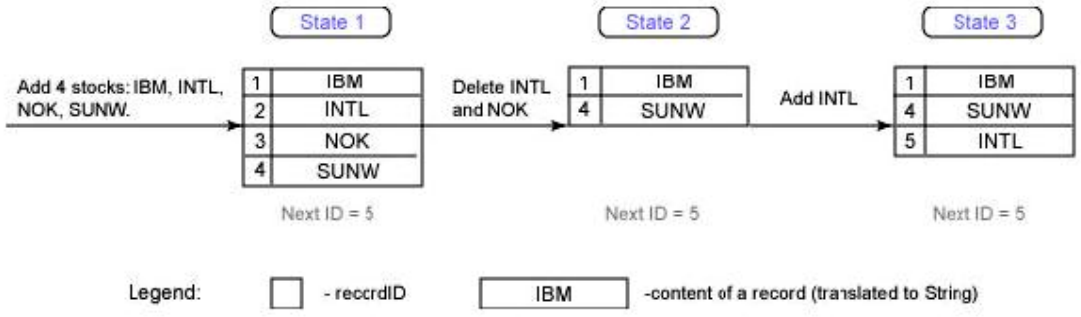

Figura. 4.15. Control del ID de los registros de un RecordStore

Además del nombre un "RecordStore" tiene otros dos atributos, que son:

[8] MIDlet suite: Los MIDlets son empaquetados y distribuidos a través de MIDlet suites. Un MIDlet suite puede contener más de un MIDlet y está formado por dos archivos: el Java Application Descriptor (.jad) y el Java Archive (.jar).

- **Versión:** Es un valor entero que se actualiza conforme vayamos insertando, modificando o borrando registros en el **RecordStore**. Este valor puede ser consultado invocando al método **RecordStore.getVersion()**.
- **Marca Temporal:** Es un entero de tipo long que representa el número de milisegundos desde el 1 de enero de 1970 hasta el momento de realizar la última modificación en el **RecordStore**. Este valor lo podemos obtener invocando al método **RecordStore.getLastModified()**.

Juntando todos estos atributos, la forma de un "RecordStore" podría ser visualizada como la Figura 4.16.

Name: datosEnc	
Version: 1.0	
TimeStamp: 2909884652314	
Records:	
Record ID	**Data**
1	byte [ ] array Datos
2	byte [ ] array Datos
…	…

Figura. 4.16. Forma de un RecordStore

4.4.2.2. Operaciones con RecordStores

Luego de haber proporcionado un conocimiento básico sobre los objetos "RecordStore", en las siguientes líneas se procederá a describir las diferentes operaciones que se pueden realizar sobre estos, a través de los métodos del paquete **javax.microedition.rms.RecordStore**.

Crear y abrir un RecordStore. Esta clase no proporciona ningún constructor, pero nos ofrece los siguientes métodos (Tabla 4.12) para poder crear o en su defecto si ya existe abrir un "RecordStore".

Campo	Valor
static RecordStore openRecordStore(String recordStoreName, boolean createIfNecessary)	Abre o si es necesario crea un RecordStore.
static RecordStore openRecordStore(String recordStoreName, boolean createIfNecessary, int authmode, boolean writable)	Abre o si es necesario crea un RecordStore que puede ser compartido con otros MIDlet suites.
static RecordStore openRecordStore(String recordStoreName, String vendorName, String suiteName)	Abre o si es necesario crea un RecordStore asociado con el MIDlet suite en cuestión.

Tabla. 4.12. Métodos para crear y abrir un RecordStore

Para crear y abrir un RS (**RecordStore**) se debe llamar a una de las clases anteriormente indicadas según sea la necesidad, tratando de que los nombres no se dupliquen y manejando las excepciones[9] si es que alguna es producida.

Manipulación de los registros de un RecordStore. Luego de abrir un objeto RS, se pueden añadir, insertar, borrar o recuperar registros. Para lo cual se utilizan los siguientes métodos (Tabla 4.13).

Campo	Valor
int addRecord(byte[] data, int offset, int numBytes)	Añade un nuevo registro al RecordStore (RS).
void deleteRecord(int recordId)	Borra el registro especificado.
int getNumRecords()	Retorna el número de registros existentes en el RS.
byte[] getRecord(int recordad)	Retorna una copia del dato almacenado en ese registro.
int getRecord(int recordId, byte[] buffer, int offset)	Retorna el dato almacenado en ese registro.
int getRecordSize(int recordId)	Retorna el tamaño (en bytes) de los datos disponibles del MIDlet en ese registro.
void setMode(int authmode, boolean writable)	Cambia el modo de acceso para este RS.
void setRecord(int recordId, byte[] newData, int offset, int numBytes)	Fija el dato en el registro indicado.

Tabla. 4.13. Métodos para la edición de los registros de un RS (RecordStore)

[9] Excepciones (exception): Son eventos que se producen durante la ejecución de un programa y que ocasionan que este deje de funcionar normalmente. Por lo general se trata de errores.

Cabe indicar que estos no son todos los métodos que el paquete **javax.microedition.rms.RecordStore** dispone para realizar operaciones con los registros de un RS, pero en esta tabla se presentaron los más relevantes para el desarrollo de este MIDlet. La definición completa de este paquete se encuentra en el Perfil MIDP 2.0[10]. Más adelante se explica la manera de almacenar y recuperar información de la memoria persistente del teléfono a través del RMS.

Cerrar y borrar un RecordStore. Si ya no se necesita más el RS, se lo puede cerrar. Con esto se consigue liberar todos los recursos que fueron utilizados por el mismo. Para esto se utiliza el método **closeRecordStore()**. Cabe indicar que se debe invocar a esta función un número igual de veces al que el RS fue abierto; es decir, el número igual de veces que el método **openRecordStore()** fue llamado.

Para borrar un RS se debe invocar al método **deleteRecordStore(String recordStoreName)**, eliminando de esta manera también a todos los registros que este contiene.

4.4.2.3. RmsDataManager

Para manejar todas estas acciones, necesarias para la gestión de los datos en la memoria persistente del teléfono, se implementó la clase **RmsDataManager.java**, la misma que tiene como una de sus variables miembro a un objeto de tipo **RecordStore**, a través del cual se realizan todas las operaciones necesarias para este fin.

Dentro de esta clase, se implementaron dos funciones que permiten almacenar y recuperar la información necesaria del RS, estas fueron: **addData(…)** y **readData(…)**; respectivamente. Dado que los registros del RS almacenan cualquier tipo de dato (String, int, char, etc.) en forma de arreglo de bytes, es necesaria la conversión del tipo de dato original a este tipo de arreglo. La función **addData(…)** realiza esta operación, de convertir y empaquetar los datos, a través de un objeto de tipo **DataOutputStream** y

[10] MIDP 2.0: Define un conjunto de APIs Java que permiten cubrir las necesidades de interfaz de usuario, almacenamiento persistente y conexiones de red.

ByteArrayOutputStream, como se muestra a continuación en la implementación del código.

```
public void addData(int idEncuentro_0, String pronEncuentro, boolean addDataFlag){
        byte[] registro;
        ByteArrayOutputStream baos;
        DataOutputStream dos;
        try{
            baos = new ByteArrayOutputStream();
            dos = new DataOutputStream(baos);
            dos.writeInt(idEncuentro_0+1);
            dos.writeUTF(pronEncuentro);
            dos.flush();
            registro = baos.toByteArray();
            //Almacena el dato en memoria y retorna el ID del RS
            if(addDataFlag)
                intIdRs[idEncuentro_0] = rs.addRecord(registro, 0, registro.length);
            //Reemplaza el dato almacenado
            else
                rs.setRecord(intIdRs[idEncuentro_0], registro, 0, registro.length);

            baos.close();
            dos.close();
        }catch(Exception e){
            System.out.println("Except: RmsDataManager/addData: " + e.getMessage());
        }
    }
```

Donde el `idEncuentro_0` representa el número de identificación del encuentro a ser almacenado como base 0, `pronEncuentro` es un String que representa al pronóstico efectuado utilizando el formato EQP1_EQP2_XXX y `addDataFlag` indica si se necesita adicionar un registro (`addDataFlag = true`) o insertar uno en una posición específica (`intIdRs[idEncuentro_0]`).

De igual manera a la hora de recuperar los datos, es necesario realizar la conversión de los mismos desde el arreglo de bytes al tipo de dato original, para ello la función **readData(...)** utiliza un objeto de tipo **DataInputStream** y **ByteArrayInputStream** de la siguiente manera:

```
public void readData(){
        ByteArrayInputStream bais;
        DataInputStream dis;
        byte[] registro = new byte[20];
```

```
        try{
            bais = new ByteArrayInputStream(registro);
            dis = new DataInputStream(bais);
            for(int i = 1; i<= rs.getNumRecords(); i++){
                rs.getRecord(i, registro, 0);
                result = dis.readInt()+dis.readUTF();
                bais.reset();
                totalForecast = totalForecast + (String)result + " ";
            }
            bais.close();
            dis.close();
        }catch(Exception e){
            System.out.println("Except: RmsDataManager/readData: " + e.getMessage());
        }
        registro = null;
    }
```

4.5. SERVLET

El Servlet desarrollado se encarga de atender todas las peticiones realizadas por el MIDlet (cliente), consultar la información solicitada en el servidor de base de datos MySQL y enviar esta información hacia al cliente, previamente estos datos son empaquetados en un String para su procesamiento en el MIDlet.

Cuando el MIDlet necesita recibir información, realiza una petición de tipo GET al Servlet. Mientras que por el contrario cuando necesita transmitir información, la petición es de tipo POST. Por esto, las clases **TbEncuentros.java** y **TbPositions.java** realizan peticiones de tipo GET y **RegDataUser.java** y **PaymentForm** las realizan de tipo POST como se lo explicó anteriormente.

En la siguiente parte se proporciona la información de respaldo sobre la comunicación entre el Servlet y la base de datos.

4.5.1. Comunicación entre el Servlet y la Base de Datos MySQL

La comunicación del Servlet con la base de datos se lo realiza mediante el **Java DataBase Connectivity (JDBC)**. El JDBC es un API (Application Programming Interface) que permite almacenar, recuperar y manipular información sobre una base de datos.

En el JDBC 3.0 existen dos paquetes: **java.sql** y **javax.sql**. El paquete **java.sql** es a menudo referido como el API núcleo del JDBC y proporciona las interfases y clases necesarias para realizar operaciones básicas sobre la información (datos) de la base de datos. Mientras que el paquete **javax.sql** es el API opcional del JDBC y permite realizar otro tipo de operaciones con los datos, incluyendo barrido de conexiones (connection pooling).

Los pasos para acceder a la información de la base de datos son los siguientes:

1. Cargar los drivers JDBC para bases de datos MySQL.
2. Establecer la conexión con una base de datos del servidor.
3. Ejecutar sentencias SQL sobre los datos.
4. Procesar los datos obtenidos.

1. Cargar los drivers JDBC para bases de datos MySQL. Los servidores de bases de datos utilizan protocolos propietarios de comunicación, los mismos que son particulares para cada uno de ellos. Para poder establecer la comunicación entre el Servlet y la base de datos MySQL, se utilizó el driver MySQL Connector/J. La carga de estos se lo realiza a través de la sentencia **Class.forName("JDBC.driver")**, donde el argumento representa el nombre completo de la clase de los drivers JDBC. El código de esta operación es el siguiente:

```
Class.forName("com.mysql.jdbc.Driver");
```

2. Establecer la conexión con una base de datos del servidor. Para acceder a una base de datos primero se necesita establecer una conexión con ella. En JDBC esta conexión está representada por la interfase **Connection**. Una instancia de la clase que implementa la interfase **Connection** es retornada luego de invocar al método **getConnection(...)**, perteneciente a la clase **DriverManager**. Este método es utilizado para intentar establecer una conexión con la base de datos. La sentencia de código que permite realizar esta operación es el siguiente:

```
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/betproject");
```

Con esta sentencia se intenta establecer una conexión con la base de datos `betproject` hospedada en un servidor de base de datos MySQL corriendo sobre la misma WorkStation (`localhost`) a través del puerto lógico `3306`.

3. Ejecutar sentencias SQL sobre los datos. Se pueden ejecutar sentencias SQL a través de la interfase **Statement**. Un objeto de tipo **Statement** es devuelto luego de que el método **createStatement()** de la interfase **Connection** fue invocado. Luego de haber establecido la conexión, se pueden ejecutar sentencias SQL a través de los métodos **executeQuery(String SQL)** y **executeUpdate(String SQL)**, los mismos que tienen como argumento un String que contiene la sentencia SQL, en la que no se necesita especificar el caracter o palabra con la que termina la sentencia, como por ejemplo: " ; " en MySQL.

El método **executeQuery(String SQL)** permite ejecutar la sentencia SQL SELECT. Luego de lo cual devuelve un objeto de tipo **ResultSet**, en donde se almacenan los datos devueltos por la consulta.

La información de la base de datos puede ser actualizada a través del método **executeUpdate(String SQL)**, el mismo que retorna el número de filas afectadas por la sentencia SQL que fue ejecutada. Esto se consigue a través de las sentencias SQL INSERT, UPDATE, o DELETE.

Como alternativa a estos métodos se puede utilizar el método **execute(String SQL)**, el mismo que permite ejecutar cualquier tipo de sentencia SQL. En el caso de que la sentencia haya sido SELECT este método retornará el valor "true" y los datos pueden ser recuperados a través del método **getResultSet(),** caso contrario si la sentencia ejecutada fue INSERT, UPDATE, o DELETE; retornará el valor "false" y el número de filas afectadas puede ser recuperado a través del método **getUpdateCount()**, ambos de la instancia **Statement**.

A continuación se presenta un extracto del código que permite ejecutar una sentencia SQL SELECT y luego retornar los datos obtenidos a un objeto de tipo **ResultSet**.

```
...
    Statement st = null;
    ResultSet rs = null;
...
        //Consulta de datos en la DB MYSQL
        Class.forName("com.mysql.jdbc.Driver");
        Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/betproject");
        st = con.createStatement();
        rs = st.executeQuery("SELECT * FROM encuentros");

...
```

En el Servlet desarrollado se utilizaron estos tres métodos para poder realizar consultas y actualizaciones en la base de datos.

4. Procesar los datos obtenidos. La interfase **ResultSet** almacena los datos obtenidos en una nueva tabla, donde sus elementos son el resultado de la sentencia ejecutada. Un objeto **ResultSet** mantiene un indicador apuntado a la fila de datos actual. En primera instancia este indicador se encuentra apuntando a la posición anterior a la primera fila, de manera que para acceder a estos datos se tiene que invocar en primer lugar al método **next()**.

Esta interfase tiene numerosos métodos que permiten recuperar datos de diferente tipo, como por ejemplo: String, bytes, double, float, etc. Estos métodos tienen la forma **getXYZ(...)**, en donde **XYZ** es el tipo de dato que estos métodos retornan y el argumento es generalmente el índice o el nombre de la columna. A continuación en la Tabla 4.14 se listan algunos de estos métodos que fueron más relevantes para el desarrollo del Servlet.

Campo	Valor
String getString(int columnIndex)	Retorna el valor de la columna con el índice indicado y en la fila actual como un String.
String getString(String columnName)	Retorna el valor de la columna con el nombre indicado y en la fila actual como un String.
int getInt(int columnIndex)	Retorna el valor de la columna con el índice indicado y en la fila actual como un Int.

Tabla. 4.14. Métodos del interfaz ResultSet para recuperar datos de tipo String e Int

Campo	Valor
Int getInt(String columnName)	Retorna el valor de la columna con el nombre indicado y en la fila actual como un Int.

Métodos del interfaz ResultSet para recuperar datos de tipo String e Int (*ContinuaciónTabla 4.14*)

El Servlet realiza varios tipos de operaciones sobre la base de datos, dependiendo del tipo de requerimiento o información que el MIDlet necesite. Para realizar estas operaciones se implementaron varios métodos que realizan distintas transacciones con diferente fin, pero los dos primeros pasos: "Carga los drivers JDBC" y "Establecer la conexión"; es común para todos. Por esto no se puede presentar el código de todos ellos, pero a continuación se lista un extracto del código utilizado para consultar en la base de datos las posiciones de los equipos y posteriormente enviar esta información al MIDlet. Esto se lo hace con el fin de exponer todos los pasos utilizados para realizar operaciones con bases de datos MySQL.

```
...
    String positions = "";
    Statement st = null;
    ResultSet rs = null;
    try{
        //Consulta de datos en la DB MYSQL

//Paso1. Carga el driver MySQL Connector/J
        Class.forName("com.mysql.jdbc.Driver");
//Paso 2. Establece la conexión a la base de datos "betproject"
        Connection                                  con                                  =
DriverManager.getConnection("jdbc:mysql://localhost:3306/betproject");
//Paso3. Crea un objeto de tipo "Statement" para ejecutar sentencias SQL
        st = con.createStatement();
//Se ejecuta la sentencia SQL y Paso 4. Procesa los datos obtenidos y almacenados en el
//objeto "ResultSet"
        rs = st.executeQuery("SELECT * FROM tab_positions");
        while(rs.next())
            positions = positions + rs.getInt("item")+": "+rs.getString("equipo")
            +":Pts="+rs.getInt("puntos")+",Dif="+rs.getInt("g_dif")+"\r\n";

        out.print(positions);
    } catch (Exception e) {
        System.out.println("Except: doPost/try query DB: " + e.getMessage());
    }finally{
        if(rs != null){
            try{
                rs.close();
```

```
            }catch(Exception e){}
            rs = null;
        }
        if(st != null){
            try{
                st.close();
            }catch(Exception e){}
            st = null;
        }
    }
...
```

El código del Servlet desarrollado (**BetProjectServlet.java**), junto con las funciones implementadas para realizar todas las transacciones con la base de datos MySQL se adjunta como anexo. La definición de los paquetes **java.sql** y **javax.sql** se encuentra en la Especificación de APIs de la Plataforma Java 2 Stantard Edition[11], v 1.4.2.

4.5.2. Descripción de la Base de Datos

En la base de datos se almacena toda la información referente al sistema de apuestas; es decir, aquí es donde se guardan y gestionan los encuentros disponibles, las tablas de posiciones, información de los usuarios y sus pronósticos.

La base de datos implementada, **betproject**, consta de cuatro tablas y su descripción se presenta a continuación:

1. Tabla "encuentros". Almacena los encuentros disponibles y que son presentados al usuario para efectuar los pronósticos. El número de encuentros disponibles está limitado a un número de 5 y en la Tabla 4.15 se presenta la descripción de los 3 campos que tiene la tabla.

[11] La especificación de APIs de la Plataforma Java 2 Stantard Edition , v 1.4.2 se puede encontrar en la siguiente URL: http://java.sun.com/j2se/1.4.2/docs/api/index.html

Campo	Tipo	Descripción
item	int(1)	Representa el item del encuentro; es decir, el orden.
equipo_a	varchar(10)	Representa el equipo local que participa en el encuentro.
equipo_b	varchar(10)	Representa el equipo visitante.

Tabla. 4.15. Descripción de la Tabla "encuentros"

2. Tabla "tab_positions". Almacena la tabla de posiciones de los equipos involucrados, en orden descendente. Las posiciones obedecen a los puntos y al gol diferencia que poseen los equipos. A continuación se presenta su descripción.

Campo	Tipo	Descripción
item	int(2)	Representa el item del encuentro; es decir, el orden.
equipo	varchar(4)	Representa el equipo en cuestión.
puntos	int(2)	Representa el puntaje alcanzado por el equipo.
g_dif	int(2)	Representa el gol diferencia del equipo.

Tabla. 4.16. Descripción de la Tabla "tab_positions"

3. Tabla "usersinfo". Almacena información referente a la identidad de los usuarios. Por razones de seguridad, en esta tabla no se incluyen muchos datos de los usuarios y el "password" se encuentra codificado mediante el algoritmo de codificación de datos **Base64**.

Campo	Tipo	Descripción
id	int(11)	Es un número secuencial, que representa la cantidad de usuarios registrados.
username	varchar(10)	Representa el nombre de usuario (user).
password	varchar(10)	Representa el password del usuario, el mismo que está codificado utilizando el algoritmo de codificación de datos Base64, descrito en el RFC 3548.
e_mail	varchar(35)	Contiene la dirección de correo electrónico del usuario.

Tabla. 4.17. Descripción de la Tabla "usersinfo"

4. Tabla "forecast_users". En esta tabla se almacenan los datos de los pronósticos efectuados por el usuario, en el siguiente formato: IDEQUIPOA_EQUIPOB_XYZ. En donde: ID representa el item del encuentro, EQUIPOA y EQUIPOB corresponden a los equipos participantes en el encuentro y XYZ corresponde al pronóstico efectuado por el usuario. Este último valor (XYZ) fue definido de la siguiente manera: X representa al pronóstico para el EQUIPOA y puede ser 1 ó 0, dependiendo si gana (1) o pierde (0); Y representa la opción de empate y Z tiene el mismo significado que X, pero para el EQUIPOB, por ejemplo: XYZ = 110, esto quiere decir que el EQUIPOA puede ganar o que los participantes pueden empatar.

Campo	Tipo	Descripción
user	varchar(20)	Representa el nombre de usuario que realizó el pronóstico.
item1	varchar(15)	Representa el pronóstico para el encuentro 1.
item2	varchar(15)	Representa el pronóstico para el encuentro 2.
item3	varchar(15)	Representa el pronóstico para el encuentro 3.
item4	varchar(15)	Representa el pronóstico para el encuentro 4.
item5	varchar(15)	Representa el pronóstico para el encuentro 5.

Tabla. 4.18. Descripción de la Tabla "forecast_users"

CAPÍTULO V

5. PRUEBAS Y RESULTADOS

5.1. DESCRIPCIÓN DEL CAPÍTULO

Como primera parte, se realiza una demostración completa sobre el funcionamiento de la aplicación desarrollada, utilizando para ello uno de los dispositivos de emulación provisto por el fabricante de teléfonos móviles Sony Ericsson en su SDK Version 2.1.5.

Posteriormente se procede a evaluar el desempeño de esta aplicación mediante el Unified Testing Criteria (UTC) Ver. 1.4 para aplicaciones basadas en la tecnología Java. Este documento reúne y unifica los criterios de prueba que los principales fabricantes de teléfonos móviles establecen para que una aplicación Java pueda ser considerada apta para ponerla en producción.

Con el resultado de la evaluación realizada a través del UTC, se procede a describir todas las correcciones y mejoras realizadas a la aplicación.

5.2. DEMOSTRACIÓN DEL FUNCIONAMIENTO DE LA APLICACIÓN

Antes de presentar el funcionamiento de la aplicación, a continuación se indican las características técnicas del MIDlet y del dispositivo de emulación. La primera versión (1.0) del MIDlet tiene como nombre **BetProjectMIDlet** y la información que contiene su archivo descriptor (**.jad)** se lista en la Tabla 5.1.

Campo	Valor	Observación
MicroEdition-Profile	MIDP-2.0	-
MicroEdition-Configuration	CLDC-1.1	-
MIDlet-Version	1.0	-
MIDlet-Name	BetProject	-
MIDlet-Vendor	Dalton Gordillo	Proveedor o en su defecto el desarrollador de la aplicación.
MIDlet-Jar-URL	BetProject.jar	URL de donde se puede descargar la aplicación (archivo **.jar**).
MIDlet-Jar-Size	148120	Tamaño del archivo **.jar** en kilobytes. Este archivo es el que será instalado en el móvil.
MIDlet-1	BetProjectMIDlet, , BetProjectMIDlet.BetProjectMIDlet	Indica que la aplicación está compuesta por un sólo MIDlet, que es el MIDlet-1.
BaseURL	http://localhost:8084/betprojservlet/betproject	Campo que se utilizó para definir la primera parte del URL al momento de realizar las conexiones HTTP al servidor.

Tabla. 5.1. Contenido del archivo descriptor (.jad) del MIDlet (BetProjectMIDlet)

Para poder realizar pruebas sobre el MIDlet, durante su desarrollo, se utilizó el dispositivo de emulación F500i de Sony Ericsson, el mismo que se muestra más abajo en la Figura 5.1 junto con sus características técnicas (Tabla 5.2). Se escogió este dispositivo primordialmente porque permitía una fácil y clara navegación sobre todas y cada una de las opciones del menú y porque también cumplía con las especificaciones técnicas que el proyecto demandaba.

Figura. 5.1. Equipo F500i de Sony Ericsson

Characteristic	Value
Screen size (W x H)	128 x 160 pixels
Pixel ratio (H:W)	1:1
Colour depth	65 536 (16 bit)
Max RMS Size	Limited only by amount of available free storage. Note: JAR file download via WAP is limited to 300 kB file size.
Memory, storage	The F500 has 10 MB internal storage memory. The amount of memory available for Java applications depends on the free amount of internal memory in the mobile phone. Other content, such as pictures, video clips and themes use the same memory pool.
Java heap memory	512 kB - 1.5 MB (dynamic, depending on available memory)
Native Video RAM available to Java	Approx. max 500 kB
CLDC version	1.1

Tabla. 5.2. Características técnicas del Equipo F500i de Sony Ericsson

Characteristic	Value
MIDP version	2.0 *Supported image formats:* GIF87a, GIF89a, PNG v 1.0 (colour depth 1, 2, 4, 8, 16 bits per pixel), BMP v 3.x, WBMP level 0. *Networking:* Secure sockets, http 1.1, https. TLS 1.0 is also supported.
OTA Recommended Practice	Yes, MIDP 2.0 compliant
Signed MIDlets	Yes
TCP Sockets	Yes
UDP Sockets	Yes

Características técnicas del Equipo F500i de Sony Ericsson (*Continuación Tabla 5.2*)

Luego de haber presentado esta introducción, se procede a realizar una explicación del MIDlet desarrollado, simulando paso a paso su funcionamiento.

En primera instancia se simula la búsqueda y posterior instalación del MIDlet a través del mecanismo de aprovisionamiento **OTA (Over The Air)**. El móvil a través de este mecanismo realiza una búsqueda de las aplicaciones disponibles para descarga en Internet, luego de ello presenta el resultado de esta búsqueda para que el usuario pueda escoger y por último el móvil se encarga de descargarla e instalarla. En la Figura 5.2 se puede apreciar la etapa de búsqueda del MIDlet y en la Figura 5.3 la descarga de la aplicación para su posterior instalación.

Figura. 5.2. Búsqueda del MIDlet en Internet.

Figura. 5.3. Descarga e instalación del MIDlet

Una vez instalada y ejecutada, la aplicación presenta como pantalla inicial el menú de opciones (Figura 5.4).

Figura. 5.4. Pantalla inicial: "Menú Pronósticos"

La primera opción de este menú, "Encuentros Jornada", presenta los encuentros disponibles de la jornada (Figura 5.5) sobre los cuales se puede realizar un pronóstico. Si se produjese alguna falla en la conexión HTTP con el servidor, el MIDlet no presentará ninguna información y por consiguiente no se podrá continuar con el siguiente paso. Si la información es presentada correctamente, el usuario podrá seleccionar cualquiera de los encuentros disponibles para luego de oprimir el botón **Selec.** pasar a la pantalla donde el sistema permite efectuar el pronóstico (Figura 5.6). Cabe indicar que no es necesario realizar el pronóstico sobre todos los encuentros disponibles, sino sobre los que el usuario desee. El pronóstico se lo realiza marcando las opciones deseadas para luego proceder a guardar la información mediante el botón **Guardar**. Luego de que se haya almacenado la información en la memoria persistente del teléfono, el MIDlet imprime un mensaje indicando el éxito de la operación (Figura 5.7). Esta información puede ser posteriormente editada, para lo cual basta con ingresar a este menú y seleccionar el encuentro deseado.

El MIDlet valida que se haya marcado alguna de las opciones del encuentro seleccionado, caso contrario no permite guardar la información y en este caso presenta un mensaje de error indicando que al menos una opción debe ser marcada (Figura 5.8). Luego de la presentación de este mensaje, la aplicación retorna al "Menú Principal".

Figura. 5.5. Encuentros disponibles para el pronóstico

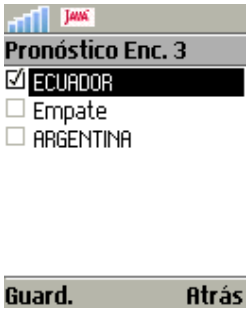

Figura. 5.6. Realización del pronóstico para el Encuentro #3

Figura. 5.7. Notificación de que la operación ha sido exitosa

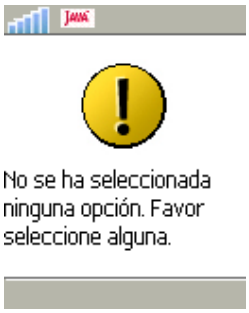

Figura. 5.8. Notificación de que no se ha realizado ningún pronóstico sobre el encuentro seleccionado

La opción **Tabla de Posiciones** presenta información sobre las posiciones de los participantes de los encuentros de la jornada. Si bien estos datos son transferidos en tiempo real, esta información corresponde a los resultados de la última fecha. El impreso de esta opción se presenta a continuación en la Figura 5.9.

Figura. 5.9. Tabla de posiciones

Para poder visualizar todos los pronósticos efectuados, así como el valor a pagar para poder participar en la apuesta, se incluyó la opción: **Mirar Pronóstico**. Dado que se tomó como base las reglas de juego e información de "Súper Golazo", la presentación de estos datos se asemeja al formato de las cartillas de este juego, como se lo puede apreciar a continuación en la Figura 5.10.

Figura. 5.10. Información de la opción "Mirar Pronóstico"

A través de la opción "Enviar Pronóstico" el usuario puede ingresar y enviar sus datos de identificación hacia el servidor para poder ser validados, como lo muestra la Figura 5.11. Si esta operación resulta ser exitosa, el MIDlet presentará un mensaje indicando esto (Figura 5.12) y pasará al siguiente punto, "Datos Forma de Pago" (Figura 5.13). En esta última parte el usuario puede seleccionar e ingresar los datos referentes a la forma de pago del pronóstico. Esta opción no fue implementada en esta versión del MIDlet, únicamente se la incluyó para simular un pago electrónico.

Figura. 5.11. Registro de los datos de usuario

Figura. 5.12. Mensaje de validación exitosa de los datos de usuario en el servidor

Figura. 5.13. Ingreso de la información de pago del pronóstico

Luego de haber enviado toda la información de los pronósticos a través del botón **Enviar** de la pantalla **Datos Forma de Pago**, el MIDlet valida el éxito o fracaso de la transacción y en ambos casos presenta un mensaje (Figura 5.14 y 5.15).

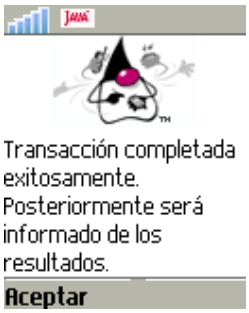

Figura. 5.14. Notificación de que los datos han sido recibidos en el servidor y actualizados en la base de datos

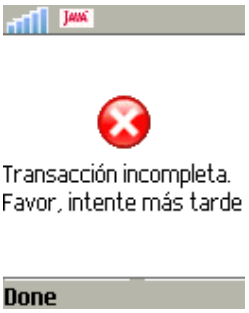

Figura. 5.15. Notificación de falla en la transacción.

El MIDlet realiza un control sobre el número de intentos erróneos que el usuario comete al momento de registrarse en el sistema a través de la pantalla “Registro Datos Usuario”. El usuario puede ingresar hasta un máximo de tres veces sus datos de identificación de manera incorrecta, en las dos primeras ocasiones el MIDlet notifica al usuario sobre este inconveniente, en la segunda a más de notificar previene de que el siguiente intento la aplicación eliminará la información almacenada sobre los pronósticos y volverá a su estado inicial como se lo muestra en las Figuras 5.16 y 5.17.

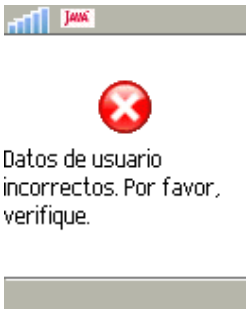

Figura. 5.16. Primer mensaje de error indicando que los datos de usuario son incorrectos

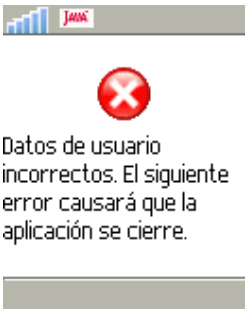

Figura. 5.17. Segunda y última notificación de error indicando que la próxima vez el sistema se cerrará.

Por último, la aplicación incorpora una opción de ayuda, en donde el usuario puede encontrar una descripción de todas las opciones que fueron implementadas en el "Menú Principal", así como también una breve explicación de las reglas del juego. Un impreso de la pantalla de esta opción se lo puede apreciar en la Figura 5.18.

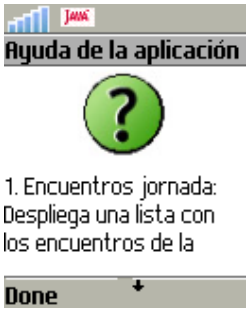

Figura. 5.18. Opción de ayuda de la aplicación

Por último, en el "Menú principal" se implementó el botón **Salir**, cuya función es liberar cualquier recurso que esté siendo utilizado en ese momento y cerrar la aplicación. Previo a esto, se presenta un mensaje de confirmación sobre esta acción.

A continuación, en la Figura 5.19, se presenta un diagrama de flujo del MIDlet.

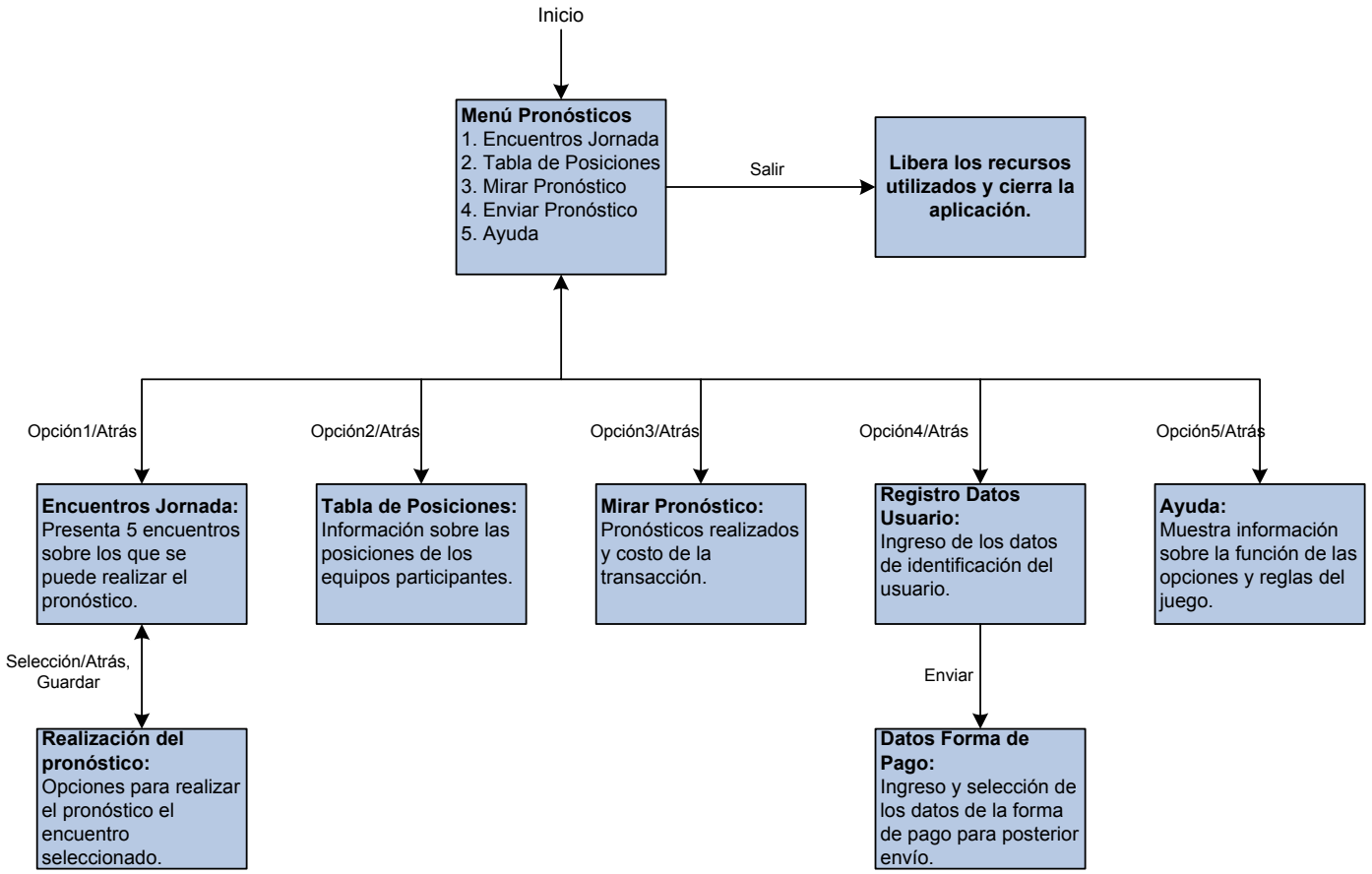

Figura. 5.19. Diagrama de flujo del MIDlet

5.3. PRUEBAS REALIZADAS AL MIDLET EN BASE AL UTC 1.4

El Unified Testing Criteria (UTC) define un criterio de pruebas para aplicaciones desarrolladas en base a la tecnología Java (TM) y fue realizado por los miembros del Unified Testing Initiative (UTI), los mismos que son: Motorola Inc., Nokia Corporation, Siemens AG mobile, Sony Ericsson Mobile Communications AB y Sun Microsystems Inc. El objetivo del UTC es asegurar que las aplicaciones Java desarrolladas para teléfonos móviles, funcionen de manera adecuada en los equipos fabricados por los integrantes del UTI, a excepción de Sun Microsystems que no se dedica a ese negocio.

Este criterio de pruebas está dividido en las siguientes categorías:

Inicio de la Aplicación (AL: Application Launch). Una vez que la aplicación se encuentra cargada en el móvil, la misma debe iniciar y detenerse correctamente en relación al dispositivo y a otras aplicaciones del dispositivo.

Interfaz de Usuario (UI: User Interface). La intención de este apartado no es definir como diseñar la interfaz de usuario sino proveer de una norma general a seguir para hacerlo. Los publicistas y las operadoras celulares definen el interfaz de usuario de acuerdo a sus requerimientos y/o lineamientos.

Funcionalidad (FN: Funcionality). La aplicación debe implementar las funcionalidades básicas y que estas sean documentadas. Fuentes de esta información son: manuales de usuario, documentación online, etc.

Operabilidad (OP: Operation). Cuando una aplicación inicia, este conjunto de pruebas ayudan a determinar de qué manera la aplicación interactúa con los otros componentes del dispositivo (hardware y software).

Seguridad (SE: Security). Las aplicaciones que pueden establecer conexiones de red son probadas en lo referente a la transmisión segura de la información y al manejo de errores que pudieren producirse en la red.

Red (NE: Network). Si una aplicación puede establecer conexiones de red, entonces debe demostrar que tiene la capacidad de comunicarse correctamente con la misma. Debe también tener la capacidad de manejar los problemas que pudieren darse sobre la red o en el lado del servidor.

Localización (LO: Localization). Las aplicaciones que serán utilizadas fuera del país donde estas fueron desarrolladas, deben permitir el cambio de lenguaje, alfabeto, formatos de fecha y hora, etc.

Una aplicación Java debe pasar exitosamente todos estos criterios de prueba para poder ser considerada apta para su utilización en un equipo celular. Todas las pruebas tienen igual peso, de manera que no se necesita establecer o definir puntaje alguno.

El último criterio, **Localización**, no será utilizado. Por cuanto la aplicación fue desarrollada teniendo en mente que su producción sería en el mercado ecuatoriano.

A continuación se presentan las pruebas realizadas al MIDlet en base a estos criterios, para posteriormente en base a los resultados obtenidos, realizar las implementaciones y correcciones necesarias.

5.3.1. Inicialización de la Aplicación (AL: Application Launch)

Criterio de prueba	Cumple	
	SI	NO
AL1. La aplicación debe ser instalada a través de OTA.	X	
AL2. La aplicación debe iniciar y cerrarse apropiadamente.	X	
AL3. La aplicación debe iniciar en no más de 15 segundos. Consideración cumplida también por aplicaciones sujetas a Digital Rights Management u otros tipos de verificación.	X	

Tabla. 5.3. Cuadro de pruebas sobre el Inicio de la aplicación (AL)

5.3.2. Interfaz de Usuario (UI: User Interface)

5.3.2.1. Claridad

Criterio de prueba	Cumple	
	SI	NO
UI1. Todo el contenido de la pantalla debe ser claro (ej. la pantalla no debe estar repleta de contenido) y legible, a menos que se pueda seleccionar el tipo de letra o el color del esquema.	X	
UI2. Toda pantalla debe aparecer el tiempo necesario para poder leer todo su contenido.	X	
UI3. Si se utilizan abreviaciones en el texto, las mismas deben ser entendibles. Por ejemplo: Los ítems de una lista deben ser abreviados convenientemente, pero aún así entendibles.	X	

Tabla. 5.4. Cuadro de pruebas sobre el Interfaz de Usuario (UI) – Claridad

5.3.2.2. Interacción con el usuario

Criterio de prueba	Cumple	
	SI	NO
UI4. Las principales funcionalidades de Salir, Acerca (About) y Ayuda deben ser fácilmente accesibles a través del Menú Principal.	X	
UI5. El interfaz de usuario de la aplicación debe ser consistente, ej. una serie común de acciones, secuencia de acciones, términos, vibración y sonidos, et.	X	
UI6. Cuando la aplicación utilice menús o selección de ítems, la función de selección de los ítems del menú debe ser claramente entendible por el usuario. Además, cada menú o ítem de selección debe realizar una operación válida.	X	
UI7. Las secuencias de acciones (ej. enviar un formulario) deben ser organizadas en grupos, con una parte inicial, media y final. Se debe proveer de una retroalimentación informativa luego de que la operación haya sido completada.	X	
UI8. La velocidad de la aplicación es adecuada y no compromete el uso de la misma. El desempeño de la aplicación es aceptable.	X	
UI9. Cualquier mensaje de error en la aplicación debe ser claro. Los mensajes de error deben explicar al usuario de manera clara la naturaleza del error e indicar qué acciones deben ser tomadas, cuando fuere este el caso.	X	
UI10. El usuario debe ser capaz de pausar y reiniciar la aplicación de manera fácil.		X
UI11. Las acciones deben poder ser revertidas de manera fácil o en su defecto indicar que la acción es irreversible.	X	
UI12. El número de pantallas por las cuales el usuario tiene que navegar es mínimo.	X	

Tabla. 5.5. Cuadro de pruebas sobre el Interfaz de Usuario (UI) – Interacción

Criterio de prueba	Cumple	
	SI	NO
UI13. Cualquier selección de una función diferente en la aplicación debe ser procesada en no más de 5 segundos. Dentro de 1 segundo, se debe incluir una alerta visual indicando que está por cumplirse una función. Esta indicación visual puede ser una entrada de usuario, barras de progreso, presentación de texto como "Por favor espere ...", etc.		X

Cuadro de pruebas sobre el Interfaz de Usuario (UI) – Interacción (*Continuación Tabla 5.5*)

5.3.2.3. Configuración y sonidos

Este tipo de pruebas no aplican para evaluar esta aplicación.

5.3.3. Funcionalidad (FN: Funcionality)

Criterio de prueba	Cumple	
	SI	NO
FN1. La aplicación debe hacer lo que realmente está pensado que debe realizar (como se encuentra especificado en la Ayuda).	X	
FN2. Una funcionalidad para salir de la aplicación debe constar explícitamente en la misma (ej. en el Menú Principal).	X	
FN3. Es recomendable un ítem de información Acerca (About) de la aplicación en el menú. Si es que esta opción es implementada, los datos de la misma deben ser consistentes con la información contenida en el JAD. El archivo JAD y la sección de Acerca deben contener información sobre el nombre del proveedor, el nombre y la versión del MIDlet.		X
FN4. Se debe proveer de una opción de ayuda en la aplicación. La ayuda debe contener: el objetivo de la aplicación, uso de las teclas (ej. para juegos), etc. Si el texto de la ayuda es demasiado largo, el mismo debe estar dividido en pequeñas secciones u organizado de diferente manera.	X	
FN5. La aplicación no debe causar daño a las aplicaciones del sistema o a los datos almacenados en el terminal.	X	

Tabla. 5.6. Cuadro de pruebas sobre la Funcionalidad (FN)

5.3.4. Operabilidad (OP: Operation)

Criterio de prueba	Cumple	
	SI	NO
OP1. Si una aplicación es interrumpida por eventos entrantes como lo es una llamada de voz, un mensaje de texto o la presentación de una notificación de error, entonces la aplicación debe reanudar su operación luego de que la interrupción haya terminado. Para esto el desarrollador debe recurrir a las APIs de los métodos **pause** y **continue**.		X

Tabla. 5.7. Cuadro de pruebas sobre la Operabilidad (OP)

5.3.5. Seguridad (SE: Security)

Criterio de prueba	Cumple	
	SI	NO
SE1. Encriptación de la información (si el sistema lo soporta) para enviar passwords u otros datos de caracter confidencial del usuario.	X	
SE2. Información confidencial, como datos de tarjetas de crédito, no deben ser almacenados localmente por la aplicación.	X	
SE3. La aplicación no debe mostrar los datos ingresados por el usuario y que son confidenciales, ej. passwords y pins. Sin embargo, se puede aceptar que la aplicación muestre brevemente el caracter para poder confirmar el dato ingresado y luego enmascararlo.	X	

Tabla. 5.8. Cuadro de pruebas sobre Seguridad (SE)

5.3.6. Red (NT: Network)

Criterio de prueba	Cumple	
	SI	NO
NT1. Si la aplicación puede establecer conexiones de red, debe presentar mensajes de error cuando intenta enviar/recibir datos y estos servicios no estén disponibles.	X[12]	
NT2. La aplicación debe ser capaz de manejar retardos. Así, para realizar una conexión debe esperar a la confirmación por parte del usuario.	X	
NT3. La aplicación debe manejar situaciones donde la conexión no está disponible.		X

Tabla. 5.9. Cuadro de pruebas sobre Red (NT)

[12] La aplicación cumple de manera parcial; es decir, no cumple totalmente con este requerimiento.

Criterio de prueba	Cumple	
	SI	NO
NT4. La aplicación debe ser capaz de cerrar la conexión que está utilizando antes de que la sesión termine.		X

Cuadro de pruebas sobre Red (NT) (*Continuación Tabla 5.9*)

5.4. RESULTADOS DE LAS PRUEBAS REALIZADAS

En lo referente a las pruebas realizadas acerca de la "5.3.1. Inicialización de la Aplicación (AL)", el MIDlet cumple con todos y cada uno de los ítems, de manera que en este caso no es necesaria ninguna corrección.

En el apartado **Interacción con el usuario** de la categoría "5.3.2. Interfaz de Usuario", la aplicación no cumple con los ítems **UI10** y **UI13**. El primero de ellos (UI10) tiene más relación con aplicaciones desarrolladas para juegos, de manera que las opciones de **pausa** y **reinicio** no será incluidas en el MIDlet. El ítem UI13 se considera variable, de manera que sería recomendable probar la aplicación en un dispositivo real trabajando en la red de un operador que ofrezca este servicio para poder determinar en que partes sería recomendable incluir mensajes que indiquen cuando la aplicación se encuentra realizando alguna acción que tarde considerablemente. Por esto, este último lineamiento (UI13) no será implementado.

En la categoría "5.3.3. Funcionalidad" no se cumple con el ítem **FN3**. Por ser considerada la opción "Acerca" necesaria para la aplicación, la misma sí será implementada en el MIDlet.

En lo referente al único ítem de la categoría "5.3.4. Operabilidad", se recomienda implementar esta funcionalidad en versiones futuras de esta aplicación, ya que se considera de gran importancia para el buen funcionamiento y estabilidad de la misma.

La aplicación cumple plenamente con todos los ítems de la categoría "5.3.5. Seguridad", de manera que no es necesaria realizar ninguna implementación aquí.

En la categoría "5.3.6. Red", el MIDlet desarrollado cumple de manera parcial con el ítem **NT1** y no cumple con los dos últimos (**NT3** y **NT4**). Con objeto de brindar más robustez a las conexiones de red realizadas por la aplicación, se incluirán mejoras y/o modificaciones sobre el manejo de errores en este tipo de operaciones por parte de la aplicación.

5.5. MODIFICACIONES E IMPLEMENTACIONES

Luego de haber identificado las modificaciones e implementaciones necesarias para que el MIDlet cumpla de buena manera con los criterios de prueba del UTC 1.4, a continuación se describe el trabajo realizado:

1. Para cumplir con el ítem FN3 de la categoría "5.3.3. Funcionalidad" se implementó la opción **Acerca de la aplicación** dentro del "Menú Principal", como se lo puede apreciar en la Figura 5.20.

Figura. 5.20. Pantalla de la opción "Acerca de la aplicación"

2. Con respecto al manejo de errores en las conexiones de red, se implementaron notificaciones de error (Figura 5.21) para cuando la conexión no pudiere ser establecida o los servidores no estuvieren disponibles. Con esto se da cumplimiento a los ítems NT1 y NT3 de la categoría de pruebas "5.3.6. Red". Refiriéndonos al ítem NT4, no se consideró necesaria la implementación de una funcionalidad para terminar la conexión de manera voluntaria, puesto que la aplicación realiza conexiones individuales de tipo HTTP hacia el servidor y si presentan errores, los mismos son manejados por la aplicación a través de notificaciones.

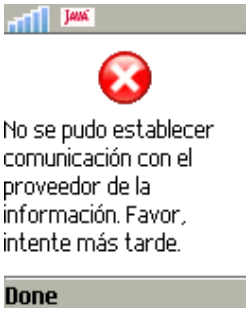

Figura. 5.21. Notificación de error en conexiones HTTP

3. Para que el usuario pudiera regresar de la pantalla "Datos Forma de Pago" hacia el "Menú Principal" para poder realizar alguna modificación en los pronósticos ya efectuados o revisar algún otro tipo de información, se implementó esta funcionalidad a través del botón (**Cancelar**). La pantalla modificada de "Datos Forma de Pago" se presenta en la Figura 5.22 y luego de que se haya seleccionado el botón **Cancelar** se presenta una notificación, como se puede apreciar en la Figura 5.23.

Figura. 5.22. Pantalla "Datos Forma de Pago" modificada (implementación de "Cancel.")

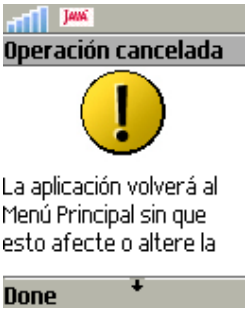

Figura. 5.23. Notificación que presenta el MIDlet luego de haber presionado el botón "Cancel." de "Datos Forma de Pago"

4. En la opción "Mirar Pronóstico" se aumento la descripción de los equipos participantes, de manera que sea más clara la información de los mismos, así como la del pronóstico efectuado. Esto se lo realizó a través de la inclusión de las etiquetas "Loc" para Locales, "Vis" para visitantes y "Emp" para indicar un empate entre estos, como se lo puede apreciar en la Figura 5.24.

Figura. 5.24. Opción "Mirar Pronóstico" modificada

Las implementaciones descritas en los puntos 3 y 4, fueron realizadas para lograr que la aplicación permita realizar modificaciones de última hora y que la información presentada sea más clara. De manera que los criterios de prueba de la categoría "5.3.2. Interfaz de Usuario (UI)" puedan ser cumplidos sobre manera.

5.6. EVALUACIÓN DE LA VERSIÓN FINAL

Luego de haber realizado las modificaciones e implementaciones necesarias, podemos concluir que la aplicación desarrollada cumple sobre manera con los objetivos propuestos al inicio del proyecto, a más de incluir otras funcionalidades para hacer que el MIDlet sea fácil y a la vez agradable al momento de ser utilizado por los usuarios. A continuación se presentan algunas de las principales ventajas y desventajas de la versión final del MIDlet:

5.6.1. Ventajas

- Las opciones del "Menú Principal" son claras y permiten realizar las operaciones que indican de manera fácil y a través de un flujo secuencial y coherente de acciones.
- La información que presenta, así como los procesos que se llevan a cabo son claros.
- Implementa opciones de Ayuda y Acerca de la aplicación, para que el usuario pueda consultar cualquier información acerca de su uso o del desarrollo del mismo.
- Realiza el manejo de errores e indica al usuario la causa y la acción o acciones a tomarse de una manera clara y poco técnica.
- La aplicación presenta el estatus o confirmación de la operación o acción que se llevó a cabo.

- Permite editar la información de una manera fácil para el usuario, de manera que no se siente obligado a realizar tareas que provocarán resultados permanentes.
- Implementa un entorno agradable y fácil de navegación a través de las diferentes opciones del menú o en las operaciones internas de las mismas.

5.6.2. Desventajas

- No presenta mensajes o notificaciones informativos cuando una transacción tomará un tiempo considerable para su ejecución. Se considera un tiempo considerable cuando la aplicación tomará más de 5 segundos en ejecutar una operación o transacción.
- No implementa un buen manejo de los eventos no predecibles, como lo son las llamadas de voz entrantes, mensajes de texto entrantes, etc.

Por último se puede concluir que la aplicación desarrollada cumple sobre manera con los objetivos propuestos para el desarrollo de la misma, a más de integrar otras funcionalidades que la hacen más asequible a la hora de se utilizada. Por otro lado, se puede apreciar que pasa la mayor parte de las pruebas del UTC 1.4 para aplicaciones profesionales basadas en Java (TM), de manera que no sería necesario realizar mayores cambios o implementaciones para poner esta aplicación en producción.

CAPÍTULO VI

6. CONCLUSIONES Y RECOMENDACIONES

6.1. CONCLUSIONES

- La versión final del sistema desarrollado (MIDlet, Servlet y Base de Datos) cumple con los objetivos planteados al inicio del proyecto, además de implementar funcionalidades que hacen al MIDlet más fácil de utilizar.

- Se utilizaron diferentes Sistemas de Emulación provistos por varios fabricantes de teléfonos celulares, como son: Sony Ericsson, Nokia y Motorola; con el fin de probar y asegurar el correcto funcionamiento del MIDlet y en consecuencia del Sistema en su totalidad (Servlet y Base de Datos). Estas pruebas fueron realizadas durante todo el desarrollo del mismo y también en la parte final, en donde se utilizó el UTC 1.4 como criterio de pruebas para verificar que la aplicación cumpla con los lineamientos y requerimientos que los principales fabricantes de teléfonos móviles exigen para considerar que la aplicación pudiera ser implementada en sus equipos.

- Dado que se tuvo que realizar un estudio y autoaprendizaje de las distintas tecnologías y plataformas utilizadas en este proyecto, ahora se tiene un grado de conocimiento y experiencia bastante bueno en lo referente a J2ME, Servlets e incluso Bases de Datos MySQL. Este último punto fue necesario debido a que se vio la necesidad de almacenar la información de una manera más idónea y con el criterio de que esta aplicación pudiera ser puesta en producción por algún operador celular o proveedor de servicios.

- Las ventajas de movilidad y capacidad de conexión que tienen los equipos de telefonía celular, así como otros dispositivos como por ejemplo las PALMs, hacen posible el desarrollo y funcionamiento de aplicaciones como esta, lo que

conlleva a la posibilidad de proveer de una amplia gama de servicios a los usuarios. Estos servicios no necesariamente tienen que ser orientados al entretenimiento sino que también pueden estar orientados a cubrir las necesidades de la vida diaria, como por ejemplo: el control domótico de una residencia, pago electrónico de algún servicio, supervisión y monitoreo de alarmas, etc.

- Una de las principales desventajas de los equipos celulares es la limitada capacidad de almacenamiento de información y de procesamiento, por lo cual los encargados de desarrollar aplicaciones para esta clase de equipos deben tener muy en cuenta estas restricciones.

6.2. RECOMENDACIONES

- Debido a que J2ME es una plataforma de última generación utilizada para el desarrollo de aplicaciones móviles y que tiene una buena acogida en el mercado de las telecomunicaciones, se recomienda la implementación total del sistema realizado en este proyecto como un nuevo tema de tesis. Por cuanto esto implica un considerable trabajo de investigación y desarrollo, así como una oportunidad de ofrecer este producto a alguna empresa para su posterior producción.

- Se recomienda implementar la opción del pago electrónico de la aplicación desarrollada (MIDlet) en este proyecto, lo que implica la inclusión y consideración de los distintos lineamientos y requerimientos que una transacción de este tipo requiere.

- También se podría dar cabida a la realización de diferentes aplicaciones utilizando la Plataforma J2ME como nuevos temas de tesis. Estas aplicaciones no necesariamente deberían estar desarrolladas con fines de entretenimiento, sino que podrían cubrir otras necesidades que tenga los usuarios, como por ejemplo: control domótico, supervisión de alarmas y especialmente se recomienda el estudio y desarrollo de los pagos electrónicos.

- En lo referente al único ítem de la categoría "5.3.4. Operabilidad", se recomienda implementar esta funcionalidad en versiones futuras de esta aplicación o de otras, ya que se considera de gran importancia para el buen funcionamiento y estabilidad de la misma.

- Para el desarrollo de aplicaciones basadas en la Plataforma J2ME se recomienda el uso de algún software dedicado para este fin, como son los Integrated Development Environment (IDE), ya que estos agrupan una serie de herramientas y utilidades que facilitan el trabajo de los desarrolladores. Para el desarrollo de este proyecto se utilizó el IDE de Sun: Sun Java Studio Mobility 6 2004Q3; que dentro de otras funcionalidades implementa un Servidor Tomcat 5.0 y conectividad con cualquier tipo de bases de datos, lo que fue de gran ayuda para el desarrollo de este proyecto.

- En esta versión de la aplicación, el costo de cada pronóstico fue fijado en $1 USD, pero se recomienda que este sea un parámetro que el servidor le entregue al usuario. Esto con el fin de que dicho valor pueda ser actualizado, de la misma forma que la información de los encuentros y la tabla de posiciones.

REFERENCIAS BIBLIOGRÁFICAS

- http://www.lcc.uma.es/~galvez/J2ME.html, J2ME
- http://java.sun.com/products/cldc/wp/KVMwp.pdf, J2ME Building Blocks for Mobile Devices
- http://www.digilife.be/quickreferences/ PT/J2ME%20Step%20by%20step.pdf, J2ME
- http://www.j2medeveloper.com/midpbook/Chapter1.pdf, Java 2 Micro Edition Basics
- http://www-128.ibm.com/developerworks/wireless/library/wi-j2me/, J2ME
- http://www.ugrad.cs.ubc.ca/~cs410/lectures/slides/cs410-J2ME-1.ppt, J2ME
- http://developers.sun.com/techtopics/mobility/getstart/articles/intro/, Introduction to Wireless Technologies
- http://www.cn-java.com/download/ data/book/j2me_stock_example.pdf, J2ME
- http://www.3gamericas.org/English/Statistics/, Estadísticas sobre usuarios de telefonía celular
- http://www.globis.ethz.ch/education/globis/ss03/foils/lect10-webarch.pdf, Servlets
- http://www.javaverified.com/index.jsp, The Java Verified Program
- http://www.javaverified.com/docs.jsp, Test Criteria & other Documents (The Java Verified Program)

- http://www4.comp.polyu.edu.hk/~csstlee/J2MELab/MIDlet-Programming-II.ppt, MIDlets

- https://www6.software.ibm.com/developerworks/education/wi-threads/, MIDlets

- http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/, Servlets

- http://www.ietf.org/rfc/rfc2617.txt, HTTP Authentication: Basic and Digest Access Authentication

- http://www.rfc-editor.org/rfcxx00.html, Official Internet Protocol Standards

- http://www.source-code.biz/snippets/java/2.htm, Base64 Encoder/Decoder

- http://www.sinotar.com/download/algorithm/README.html, Base64 Encoder/Decoder

- http://www.globis.ethz.ch/education/globis/slides/globis-2004-Web-Architecture.pdf, Servlets

- http://java.sun.com/docs/books/tutorial/information/download.html, Información variada sobre la tecnología Java

- http://java.sun.com/j2se/1.4.2/docs/api/index.html, Java(TM) 2 Platform, Standard Edition, v 1.4.2 API Specification

- http://java.sun.com/j2se/1.4.2/download.html, Descarga de J2SE v 1.4.2_06 SDK

- http://java.sun.com/products/j2mewtoolkit/download-2_1.html, Descarga de J2ME Wireless Toolkit 2.1_01

- http://wwws.sun.com/software/download/products/41085de1.html, Descarga de Sun Java(TM) Studio Mobility 6 2004Q3

- http://java.sun.com/j2se/1.5.0/download.jsp, Descarga de J2SE 5.0 JDK + JRE

- http://developer.sonyericsson.com/site/global/docstools/java/p_java.jsp, Información y descargas sobre J2ME de Sony Ericsson

- http://www.forum.nokia.com/main/1,,0333,00.html#java, Herramientas e información de soporte para desarrollar aplicaciones, para teléfonos móviles de Nokia

- http://idenphones.motorola.com/idenDeveloper/developer/developer_tools.jsp, Herramientas de desarrollo de Motorola

- http://jakarta.apache.org/tomcat/index.html, Descargas de Apache Tomcat

- http://dev.mysql.com/downloads/, Descargas de MySQL

- http://dev.mysql.com/doc/, Documentación de MySQL

ANEXOS

ANEXO No. 1: CÓDIGO FUENTE DEL MIDLET Y SUS CLASES

MIDlet: BetProjectMIDlet.java

Código Fuente:

```
/*
 * BetProjectMIDlet.java
 *
 * Created on 7 de marzo de 2005, 07:26 PM
 */

//package BetProjectMIDlet;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 *
 * @author  Dalton Gordillo
 * @version
 */
public class BetProjectMIDlet extends MIDlet implements CommandListener{
    protected Display myDisplay;    //Display
    private Command cmdExit, cmdSelect;
    private String options[] = {" Encuentros Jornada", " Tabla de Posiciones", " Mirar
Pronóstico",
    " Enviar Pronóstico"," Ayuda", " Acerca de la aplic."};  //Opciones del menu principal
    private Image[] imgs = new Image[6];
    protected List lsMainMenu;  //Form principal
    private TbEncuentros mainTabEnc;    //Tabla de Encuentros de la Jornada
    private TbPositions tabPositions;    //Tabla de Posiciones
    private RegDataUser rgDataUser; //Maneja los datos de usuario
    private ForecastViewer forecastView;    //Resumen del pronóstico y su costo
    private final int IMPLICIT_LIST = 3;
    private final int ALERT_DELAY = 1200;
    protected RmsDataManager rsdatosEnc;  //Gestionará el almacen./edición de los
pronósticos a través del RMS
    private Alert alrNoData;
    private ConfirmExit alertExit;

    public class ConfirmExit extends Alert implements CommandListener{
        BetProjectMIDlet midlet;
        Displayable parent;
        Command cmdOk, cmdCancel;

        public ConfirmExit(String title, BetProjectMIDlet midlet, Displayable parent){
            super(title);
            this.midlet = midlet;
            this.parent = parent;

            Image img = null;
            try{
                img = Image.createImage("/help2.png");
                setImage(img);
            }catch(Exception e){
                System.out.println("Except: BetProjectMIDlet/ConfirmExit/try del img: " +
e.getMessage());
            }
            setString("Está seguro que desea abandonar la aplicación ?");
            setTimeout(Alert.FOREVER);
           midlet.myDisplay.setCurrent(this);

            cmdOk = new Command("Aceptar", Command.OK, 1);
            cmdCancel = new Command("Cancel.", Command.EXIT, 1);

            addCommand(cmdOk);
            addCommand(cmdCancel);
            setCommandListener(this);
        }

        public void commandAction(Command cmd, Displayable dis){
            if(cmd == cmdOk){
                midlet.destroyApp(false);
                midlet.notifyDestroyed();
            }
```

```
            else if(cmd == cmdCancel){
                midlet.myDisplay.setCurrent(parent);
            }
        }
    }

    //Constructor
    public BetProjectMIDlet(){
        myDisplay = Display.getDisplay(this);

        cmdExit = new Command("Salir", Command.EXIT, 1);
        cmdSelect = new Command("Selec.", Command.ITEM, 1);
        rsdatosEnc = new RmsDataManager("datosEnc");
        alrNoData = new Alert(null);

        try{
        /*Enc*/imgs[0] = Image.createImage("/encuentros.png");
        /*Pos*/imgs[1] = Image.createImage("/positions.png");
        /*For*/imgs[2] = Image.createImage("/forecast_view.png");
        /*Reg*/imgs[3] = Image.createImage("/indicator.png");
        /*Help*/imgs[4] = Image.createImage("/help.png");
        /*About*/imgs[5] = Image.createImage("/about.png");
        }catch(Exception e){
            System.out.println("Except: BetProjectMIDlet/Constructor/try imgs: " +
e.getMessage());
        }

        lsMainMenu = new List("Menú Pronósticos", List.IMPLICIT, options, imgs);

        lsMainMenu.addCommand(cmdExit);
        lsMainMenu.addCommand(cmdSelect);

        lsMainMenu.setCommandListener(this);
    }

    public void startApp() {
        myDisplay.setCurrent(lsMainMenu);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
       rsdatosEnc.closeRecStore();
       rsdatosEnc.destroyRecStore();
       System.out.println("El RecordStore ha sido cerrado y borrado.");
    }

    public void showAlert(){
        //Presenta Alert de almacenamiento de info.
        Image img = null;
        try{
            img = Image.createImage("/alert.png");
        }catch(Exception e){
            System.out.println("Except: BetProjectMIDlet/showAlert/try del img: " +
e.getMessage());
        }
        alrNoData.setImage(img);
        alrNoData.setString("No existe información almacenada al momento.");
        alrNoData.setTimeout(ALERT_DELAY);
        myDisplay.setCurrent(alrNoData, lsMainMenu);
    }

    public void commandAction(Command cmd, Displayable dis){
        if(cmd == cmdExit){
            alertExit = new ConfirmExit("Confirmación", this, lsMainMenu);
            myDisplay.setCurrent(alertExit);
        }
        //Opciones del Menu Principal
        else if(cmd == cmdSelect || cmd == List.SELECT_COMMAND){
            switch(lsMainMenu.getSelectedIndex()){
            case 0: //Encuentros Jornada
                mainTabEnc = new TbEncuentros("Encuentros Jornada", IMPLICIT_LIST, this,
lsMainMenu);
                myDisplay.setCurrent(mainTabEnc);
                break;
```

```
            case 1: //Tabla de posiciones
                tabPositions = new TbPositions("Tabla de Posiciones", this, lsMainMenu);
                myDisplay.setCurrent(tabPositions);
                break;

            case 2: //Mirar Pronóstico
                forecastView = new ForecastViewer("Pronóstico y costo", this, lsMainMenu);
                try{
                    if(rsdatosEnc.rs.getNumRecords() > 0)
                        myDisplay.setCurrent(forecastView);
                    else
                        showAlert();
                }catch(Exception e){}

                break;

            case 3: //Enviar datos de la apuesta
                rgDataUser = new RegDataUser("Registro Datos Usuario", this, lsMainMenu);
                try{
                    if(rsdatosEnc.rs.getNumRecords() > 0)
                        myDisplay.setCurrent(rgDataUser);
                    else
                        showAlert();
                }catch(Exception e){}

                break;

            case 4: //Ayuda de la aplicacion
                Image img = null;
                try{
                        img = Image.createImage("/help2.png");
                }catch(Exception e){
                    System.out.println("Except: BetProjectMIDlet/Help/try del img: " +
e.getMessage());
                }

                Alert appHelp = new Alert("Ayuda de la aplicación", "1. Encuentros jornada:
Despliega una lista" +
                " con los encuentros de la jornada, para luego de la selección proceder a
realizar el pronóstico.\n"+
                "2. Tabla de posiciones: Muestra la tabla de posiciones de los equipos.\n
3. Mirar Pronóstico: Permite " +
                "visualizar el pronóstico de los encuentros seleccionados, así como el
valor de la apuesta. Se coloca una [X]" +
                " o un [ ] para identificar el equipo ganador o perdedor, además se puede
identificar si el participante es" +
                " Local (Loc), Visitante (Vis) o si hay un empate (Emp). Por cada " +
                "opción seleccionada ([X]) el monto a cobrar será de 1 USD, la aplicación
no permite realizar/guardar un " +
                "pronóstico sin que al menos una de las opciones esté marcada." +
                "\n 4. Registro de datos: Envía la información de la apuesta y la forma de
pago previo registro del usuario." +
                "\n 5. Ayuda: Presenta la información de ayuda.\n 6. Acerca de la aplic.:
Presenta información " +
                "sobre la aplicación desarrollada, como el nombre, versión, etc." , img,
null);
                appHelp.setTimeout(Alert.FOREVER);
                myDisplay.setCurrent(appHelp);
                break;

            case 5: //About
                Image imgAbout = null;
                try{
                        imgAbout = Image.createImage("/signature.PNG");
                }catch(Exception e){
                    System.out.println("Except: BetProjectMIDlet/Help/try del imgAbout: " +
e.getMessage());
                }

                Alert appAbout = new Alert("Acerca de la aplicación", "   " +
                this.getAppProperty("MIDlet-Name") + " Ver. "+this.getAppProperty("MIDlet-
Version")+
                "\nDesarrollado por: \n" + "   " + this.getAppProperty("MIDlet-Vendor") +
                "\nReservados todos los Derechos - 2005", imgAbout, null);
                appAbout.setTimeout(Alert.FOREVER);
                myDisplay.setCurrent(appAbout);
                break;
```

```
                }
            }
        }
}
```

Clase: ForecastViewer.java

Código Fuente:

```
/*
 * SubForm.java
 *
 * Created on 7 de febrero de 2005, 11:58 PM
 */

//package BetProjectMIDlet;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import java.io.*;
import javax.microedition.rms.*;
import java.util.Vector;

/**
 *
 * @author  Dalton Gordillo
 * @version
 */
public class ForecastViewer extends SubForm {//implements CommandListener,
ItemStateListener {
    private Vector datapartSp = new Vector();
    int cost = 0;
    //Constructor
    public ForecastViewer(String title, BetProjectMIDlet midlet, Displayable parent){
        super(title, midlet, parent);
        showData();

    }

    public void showData(){
        try{
            int regs = midlet.rsdatosEnc.rs.getNumRecords();
            if(regs > 0){
                //Presenta los pronósticos almacenados
                midlet.rsdatosEnc.resetTotalForecast();
                midlet.rsdatosEnc.readData();
                String forecast = midlet.rsdatosEnc.totalForecast;
                splitData(forecast, regs);
                if(!datapartSp.isEmpty()){
                    append("    Loc      Emp      Vis");
                    for(int i=0; i <datapartSp.size(); i++)
                        append((String)datapartSp.elementAt(i));

                    String auxCost = "";
                    append(new StringItem("Costo: $ ", auxCost.valueOf(cost)));
                }
            }
        }catch(Exception e){
            System.out.println("Except: ForecastViewer/showData: " + e.getMessage());
        }
    }

    public void splitData(String forecast, int regs){
        int j = 0;
        int k = 0;
        String[] flags = {"", "", ""};
        boolean aux = true;
        String auxSp = null;
        String dataForecast = "";

        for(int i=0; i<regs; i++){

            flags[0] = forecast.substring(k+9, k+10).equals("1")?"[X]":"[  ]";
            flags[1] = forecast.substring(k+10, k+11).equals("1")?"[X]":"[  ]";
            flags[2] = forecast.substring(k+11, k+12).equals("1")?"[X]":"[  ]";

            if(forecast.substring(k+9, k+10).equals("1"))
                cost++;
```

```
            if(forecast.substring(k+10, k+11).equals("1"))
                cost++;
            if(forecast.substring(k+11, k+12).equals("1"))
                cost++;

            datapartSp.addElement(forecast.substring(j*13, j*13+1) + ": " +
            flags[0] + " " + forecast.substring(k+1, k+4) + "  " + flags[1] + "  " +
            forecast.substring(k+5, k+8) + " " + flags[2] + " ");

            k+=13;
            j+=1;
        }
    }

    public void commandAction(Command command, Displayable displayable) {
         super.commandAction(command, displayable);
    }
}
```

Clase: RegDataUser.java

Código Fuente:

```
/*
 * RegDataUser.java
 *
 * Created on 21 de marzo de 2005, 10:06 PM
 */

//package BetProjectMIDlet;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.*;
import java.util.Vector;

/**
 *
 * @author  Dalton Gordillo
 * @version
 */
public class RegDataUser extends SubForm implements Runnable{
    private TextField txtfUser; //Campo para ingresar el user
    private TextField txtfPsw;  //Campo para ingresar el password
    private Command cmdOk;  //Acepta datos ingresados
    private String user = null; //Almacena datos de usuario
    private String psw = null;
    Alert alInvUsrData;   //Indica que algún dato del usuario es incorrecto
    Alert alUsrDataOK;    //Datos de usuario OK
    Alert alAppBlock;    //Bloqueo de la aplicación
    Alert alTransacOK;
    private int errCount = 0;   //Controla el # de intentos erróneos de logearse
    private final int SC_OK = 200;
    private final int SC_UNAUTHORIZED = 401;
    private final int ALERT_DELAY = 1200;
    private PaymentForm paymForm;   //Form que maneja el pago del pronósitco
    private String strResp = null;

    public class PaymentForm extends SubForm implements Runnable{
        private Command cmdSubmit;
        private Command cmdCancel;
        private BetProjectMIDlet midlet;

        private TextField txtfCardNumber = null;
        private TextField txtfExpDate = null;
        private ChoiceGroup choiceCardType = null;
        private String cardOptions[] = {" MasterCard", " Visa", " AMEX"};

        public PaymentForm(String title, BetProjectMIDlet midlet, Displayable parent){
            super(title, midlet, parent);

            this.midlet = midlet;
            cmdCancel = new Command("Cancel.", Command.SCREEN, 1);
            cmdSubmit = new Command("Enviar", Command.SCREEN, 0);
            txtfCardNumber = new TextField("No. ", null, 10, TextField.NUMERIC);
            txtfExpDate = new TextField("Vence:", null, 6, TextField.NUMERIC);
            choiceCardType = new ChoiceGroup("Proveedor Tarjeta: ", ChoiceGroup.EXCLUSIVE,
cardOptions, null);

            append(choiceCardType);
            append(new StringItem("Datos Tarjeta: ", null));
            append(txtfCardNumber);
            append(txtfExpDate);
            addCommand(cmdSubmit);
            addCommand(cmdCancel);
            removeCommand(backCmd);
        }

public void run() {
    HttpConnection con=null;
    OutputStream os=null;
    InputStream is=null;
```

```
    try{
        String url = midlet.getAppProperty("BaseURL") + "?queryStr=DataForescast";
        con = (HttpConnection)Connector.open(url);
        con.setRequestMethod(HttpConnection.POST);
        con.setRequestProperty("Content-Language", "es-ES");
        con.setRequestProperty("Content-Type", "text/plain");
        con.setRequestProperty("Accept", "text/plain");
        con.setRequestProperty("User-Agent", "Profile/MIDP-2.0 Configuration/CLDC-1.0");
        con.setRequestProperty("Connection", "Close");

        midlet.rsdatosEnc.resetTotalForecast();
        midlet.rsdatosEnc.readData();
        os = con.openOutputStream();
        os.write(midlet.rsdatosEnc.totalForecast.getBytes());
        os.close();
        os = null;

        is = con.openInputStream();
        procesarRespuesta(con, is);

        if(strResp.equals("1"))
            //Presenta mensaje de transacción completa
            showConfirm(true);
        else
            //Presenta mensaje de error en la transacción
            showConfirm(false);

    }catch(Exception e){
        System.out.println("Except: PaymentForm/run/try: " + e.getMessage());
        //Presenta mensaje de error en la transacción
        showConfirm(false);


    }finally {
     try{
         if (con != null) con.close();
     }catch(Exception e){
         System.out.println("Except: PaymentForm/run/finally/con: " + e.getMessage());
     }
    }
}

        public void procesarRespuesta(HttpConnection c, InputStream is)
        throws IOException{
            if(c.getResponseCode() == HttpConnection.HTTP_OK){
               int len = (int)c.getLength();
               if(len > 0 ){
                 int actual = 0;
                 int bytesread = 0 ;
                 byte[] data = new byte[len];

                 while ((bytesread != len) && (actual != -1)) {
                    actual = is.read(data, bytesread, len - bytesread);
                    bytesread += actual;
                    strResp = new String(data);
                 }
               }
                else{
                    ByteArrayOutputStream baos = new ByteArrayOutputStream();
                    int ch;
                    while((ch = is.read()) != -1)
                        baos.write(ch);
                    strResp = new String(baos.toByteArray());
                    baos.close();
                }
            }
            else{
                System.out.println("Error: "+c.getResponseMessage());
            }
        }

        public void showConfirm(boolean flag){
            if(flag){
                Image img = null;
                try{
                    img = Image.createImage("/Duke.png");
                    alTransacOK.setImage(img);
```

```
                }catch(Exception e){
                    System.out.println("Except: RegDataUser/showConfirm/try img: " +
e.getMessage());
                }
                alTransacOK.setString("Transacción completada exitosamente. Posteriormente
" +
                "será informado de los resultados.");
                alTransacOK.setTimeout(Alert.FOREVER);
                Command cmdAccept = new Command("Aceptar", Command.SCREEN, 1);
                alTransacOK.addCommand(cmdAccept);
                alTransacOK.setCommandListener(this);

                midlet.myDisplay.setCurrent(alTransacOK, this);
            }
            else{
                Image img = null;
                try{
                    img = Image.createImage("/error2.png");
                    alTransacOK.setImage(img);
                }catch(Exception e){
                    System.out.println("Except: RegDataUser/showConfirm/try img: " +
e.getMessage());
                }
                alTransacOK.setString("Transacción incompleta. Favor, intente más tarde.");
                alTransacOK.setTimeout(Alert.FOREVER);
                midlet.myDisplay.setCurrent(alTransacOK, this);
            }
        }

        public void commandAction(Command cmd, Displayable dis){
            super.commandAction(cmd, dis);
            if(cmd == cmdSubmit){
                Thread t = new Thread(this);    //Crea un thread para la conexión
                t.start();
            }
            else if(cmd == cmdCancel){
                Alert alrCancel = new Alert("Operación cancelada");
                Image img = null;
                try{
                    img = Image.createImage("/alert.png");
                }catch(Exception e){
                    System.out.println("Except: BetProjectMIDlet/PaymentForm/try del img: "
+ e.getMessage());
                }
                alrCancel.setImage(img);
                alrCancel.setString("La aplicación volverá al Menú Principal sin que esto "
+
                "afecte o altere la información almacenada.");
                alrCancel.setTimeout(ALERT_DELAY+600);
                midlet.myDisplay.setCurrent(alrCancel, midlet.lsMainMenu);
            }
            else if(dis == alTransacOK){
                midlet.destroyApp(false);
                midlet.notifyDestroyed();
            }
        }
    }

    //Constructor
    public RegDataUser(String title, BetProjectMIDlet midlet, Displayable parent){
        super(title, midlet, parent);
        txtfUser = new TextField("User: ", "", 10,
TextField.SENSITIVE|TextField.NON_PREDICTIVE);
        txtfPsw = new TextField("Psw: ", "", 6, TextField.PASSWORD|TextField.NUMERIC);
        cmdOk = new Command("Enviar", Command.SCREEN, 1);
        alInvUsrData = new Alert(null);
        alUsrDataOK = new Alert(null);
        alAppBlock = new Alert(null);
        alTransacOK = new Alert(null);

        append(txtfUser);
        append(txtfPsw);
        addCommand(cmdOk);
    }

    public void run() {
        HttpConnection con=null;
```

```
            try{
            //Estableciendo parametros para la conexion
            String url = midlet.getAppProperty("BaseURL") + "?queryStr=UserValidation";
            con = (HttpConnection)Connector.open(url);
            con.setRequestMethod(HttpConnection.POST);
            con.setRequestProperty("Content-Language", "es-ES");
            con.setRequestProperty("Content-Type", "text/plain");
            con.setRequestProperty("Accept", "text/plain");
            con.setRequestProperty("User-Agent", "Profile/MIDP-2.0 Configuration/CLDC-
1.0");
            con.setRequestProperty("Connection", "Close");
            con.setRequestProperty("Authorization", "Basic " +
                                   Base64Coder.encode(user.toLowerCase() + ":" + psw));
            int responseCod = con.getResponseCode();
            if(responseCod == SC_OK){
                errCount = 0;
                //Presentación de mensaje de aceptación
                //Load de la imagen
                Image imgError = null;
                try{
                    imgError = Image.createImage("/login_OK.png");
                }catch(Exception e){
                    System.out.println("Except: RegDataUser/run/try imgError/conOK: " +
e.getMessage());
                }
                alUsrDataOK.setImage(imgError);
                alUsrDataOK.setString("Datos de usuario correctos.");
                alUsrDataOK.setTimeout(1200);

                paymForm = new PaymentForm("Datos Forma de Pago", midlet, this);
                midlet.myDisplay.setCurrent(alUsrDataOK, paymForm);
            }
            else if(responseCod == SC_UNAUTHORIZED){
                //Presenta error
                //System.out.println(errCount);
                if(errCount == 0){
                    txtfUser.setString("");
                    txtfPsw.setString("");
                    dispFormError(false);
                }
                else if(errCount == 1){
                    txtfUser.setString("");
                    txtfPsw.setString("");
                    dispFormError(true);
                }
                else{
                    midlet.destroyApp(false);
                    midlet.notifyDestroyed();
                }
                errCount++;
            }
            }catch(Exception e){
                System.out.println("Except: RegDataUser/run/try: " + e.getMessage());
            }finally {
             try{
                 if (con != null) con.close();
              }catch(Exception e){
                  System.out.println("Except: RegDataUser/run/finally/con: " +
e.getMessage());
              }
        }
    }

    //Presenta el form indicando un error en los datos ingresados por el usuario
    //y si es la 3ra. opción destruye el MIDlet.
    void dispFormError(boolean lastChance){
        if(!lastChance){
            Image imgError = null;
            try{
                imgError = Image.createImage("/error2.png");
                alInvUsrData.setImage(imgError);
            }catch(Exception e){
                System.out.println("Except: RegDataUser/dispFormError/try imgError: " +
e.getMessage());
            }
            alInvUsrData.setString("Datos de usuario incorrectos. Por favor, verifique. ");
```

```
            alInvUsrData.setTimeout(1200);
            midlet.myDisplay.setCurrent(alInvUsrData, this);
        }
        else{
            Image imgError = null;
            try{
                imgError = Image.createImage("/error2.png");
            }catch(Exception e){
                System.out.println("Except: RegDataUser/run/blockApp: " + e.getMessage());
            }
            alAppBlock.setImage(imgError);
            alAppBlock.setString("Datos de usuario incorrectos. El siguiente error causará"
+
            " que la aplicación se cierre.");
            alAppBlock.setTimeout(2000);
            midlet.myDisplay.setCurrent(alAppBlock, this);
        }
    }

    public void commandAction(Command cmd, Displayable dis){
        super.commandAction(cmd, dis);

        if(cmd == cmdOk){
            user = txtfUser.getString();
            psw = txtfPsw.getString();

            if(!(user.equals("")) && !(psw.equals(""))){
                Thread t = new Thread(this);    //Crea un thread para la conexión
                t.start();
            }
            else{
                //Presenta error
                dispFormError(false);
            }
        }
    }
}
```

Clase: RmsDataManager.java

Código Fuente:

```
/*
 * RmsDataManager.java
 *
 * Created on 11 de abril de 2005, 12:14 PM
 */

//package BetProjectMIDlet;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;
import javax.microedition.rms.*;

/**
 *
 * @author  Dalton Gordillo
 */
public class RmsDataManager{
    protected RecordStore rs;
    private String result;  //Almacena la información extraída del registro
    int cont = 0;
    String rsName;      //Nombre del RecordStore
    int[] intIdRs = {0, 0, 0, 0, 0}; //Almacena el Id del RS donde se almacenó el # de
encuentro,
                                //inicia con cero
    private boolean[] selectedOpts = new boolean[3];    //Almacena las opciones
seleccionadas
                                                        //sobre el encuentro
    protected String totalForecast = ""; //Alamacena todo el pronóstico en un String

    /** Creates a new instance of RmsDataManager */
    public RmsDataManager(String rsName) {
        try{
            this.rsName = rsName;
            rs = RecordStore.openRecordStore( rsName, true);
        }catch(Exception e){
            System.out.println("Except: RmsDataManager/const: " + e.getMessage());
        }
    }
    //Añade o reemplaza un registro
    public void addData(int idEncuentro_0, String pronEncuentro, boolean addDataFlag){
        byte[] registro;
        ByteArrayOutputStream baos;
        DataOutputStream dos;
        try{
            baos = new ByteArrayOutputStream();
            dos = new DataOutputStream(baos);
            dos.writeInt(idEncuentro_0+1);
            dos.writeUTF(pronEncuentro);
            dos.flush();
            registro = baos.toByteArray();
            //Almacena el dato en memoria y retorna el ID del RS
            if(addDataFlag)
                intIdRs[idEncuentro_0] = rs.addRecord(registro, 0, registro.length);
            //Reemplaza el dato almacenado
            else
                rs.setRecord(intIdRs[idEncuentro_0], registro, 0, registro.length);

            baos.close();
            dos.close();
        }catch(Exception e){
            System.out.println("Except: RmsDataManager/addData: " + e.getMessage());
        }
    }
    //Recupera todos los registros
    public void readData(){
        ByteArrayInputStream bais;
        DataInputStream dis;
        byte[] registro = new byte[20];
        try{
```

```
            bais = new ByteArrayInputStream(registro);
            dis = new DataInputStream(bais);
            for(int i = 1; i<= rs.getNumRecords(); i++){
                rs.getRecord(i, registro, 0);
                result = dis.readInt()+dis.readUTF();
                bais.reset();
                totalForecast = totalForecast + (String)result + " ";
            }
            bais.close();
            dis.close();
        }catch(Exception e){
            System.out.println("Except: RmsDataManager/readData: " + e.getMessage());
        }
        registro = null;
    }
    //Cierra el RecordStore
    public void closeRecStore(){
        try{
            rs.closeRecordStore();
        }catch(Exception e){
            System.out.println("Except: RmsDataManager/closeRecStore: " + e.getMessage());
        }
    }
    //Destruye el RecordStore
    public void destroyRecStore(){
        try{
            RecordStore.deleteRecordStore(rsName);
        }catch(Exception e){
            System.out.println("Except: RmsDataManager/destroyRecStore: " +
e.getMessage());
        }

    }

    public boolean checkDataSelectd(int selection){
    if (intIdRs[selection] != 0){
        ByteArrayInputStream bais;
        DataInputStream dis;
        byte[] registro = new byte[20];
        String strRecordVal = null;
        try{
            bais = new ByteArrayInputStream(registro);
            dis = new DataInputStream(bais);
            rs.getRecord(intIdRs[selection], registro, 0);
            strRecordVal = dis.readInt()+dis.readUTF();
            //Setea los valores del pronóstico
            getOptionsSaved(strRecordVal);
            bais.close();
            dis.close();
        }catch(Exception e){
            System.out.println("Except: RmsDataManager/checkDataSelectd: " +
e.getMessage());
        }
        registro = null;
        return true;
    }

    else{
        return false;
    }
    }
    //Recupera las opciones seleccionadas
    public void getOptionsSaved(String strRecordVal){
        int j = 9;
        String aux1 = "1";
        String aux2;
        for(int i=0; i<3; i++){
            aux2 = strRecordVal.valueOf(strRecordVal.charAt(j));
            if(aux1.equals(aux2))
                selectedOpts[i] = true;
            else
                selectedOpts[i] = false;
            j+=1;
        }
    }

    public boolean[] retSelectdFlags(){
```

```
        return selectedOpts;
    }

    public void resetTotalForecast(){
        totalForecast = "";
    }
}
```

Clase: SubForm.java

Código Fuente:

```
/*
 * SubForm.java
 *
 * Created on 7 de febrero de 2005, 11:58 PM
 */

//package BetProjectMIDlet;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

/**
 *
 * @author  Dalton Gordillo
 * @version
 */
public class SubForm extends Form implements CommandListener{//, ItemStateListener {

    Command backCmd;
    BetProjectMIDlet midlet;
    Displayable parent;

    public SubForm(String title, BetProjectMIDlet midlet, Displayable parent){
       super(title);
        this.midlet = midlet;
        this.parent = parent;

        backCmd = new Command("Atrás", Command.BACK, 1);

        addCommand(backCmd);
        setCommandListener(this);
    }

    /**
     * constructor por defecto
     */
    public SubForm() {
        super("Sample Title");
        append("Sample Text Item");
        try {
            // Set up this form to listen to command events
            setCommandListener(this);
//            // Set up this form to listen to changes in the internal state of its
interactive items
//            setItemStateListener(this);
            // Add the Exit command
            addCommand(new Command("Exit", Command.EXIT, 1));
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Called when user action should be handled
     */
    public void commandAction(Command command, Displayable displayable) {
        if(command == backCmd){
            if(parent != null)
                midlet.myDisplay.setCurrent(parent);
        }
    }
//    /**
//     * Called when internal state of any item changed
//     */
//    public void itemStateChanged(Item item) {
//    }
}
```

Clase: SubList.java

Código Fuente:

```
/*
 * SubList.java
 *
 * Created on 14 de marzo de 2005, 08:28 PM
 */

//package BetProjectMIDlet;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
/**
 *
 * @author  Dalton Gordillo
 * @version
 */
public class SubList extends List implements CommandListener {
    Command cmdBack;
    BetProjectMIDlet midlet;
    Displayable parent;

    public SubList(String title, int type, BetProjectMIDlet midlet, Displayable parent){
       super(title, type);
        this.midlet = midlet;
        this.parent = parent;

        cmdBack = new Command("Atrás", Command.BACK, 1);

        addCommand(cmdBack);
        setCommandListener(this);
    }

    public SubList() {
        super("Sample Title", List.IMPLICIT);
        append("Sample Item 1", null);
        append("Sample Item 2", null);
        try {
            // Set up this list to listen to command events
            setCommandListener(this);
            // Add the Exit command
            addCommand(new Command("Exit", Command.EXIT, 1));
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Called when action should be handled
     */
    public void commandAction(Command command, Displayable displayable) {
        if(command == cmdBack){
            if(parent != null)
                midlet.myDisplay.setCurrent(parent);
        }
    }

}
```

Clase: TbEncuentros.java

Código Fuente:

```
/*
 * SubList.java
 *
 * Created on 14 de marzo de 2005, 08:28 PM
 */

//package BetProjectMIDlet;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.*;
import java.util.Vector;
import javax.microedition.rms.*;

/**
 *
 * @author  Dalton Gordillo
 * @version
 */

//Implementa un List que muestra los Encuentros de la Jornada
public class TbEncuentros extends SubList implements Runnable{
    private String strResp = "";
    private final int MULTIPLE_LIST = 2;
    private Vector itemsEnc = new Vector();     //Almacena los equipos de c/encuentro
    private Vector datapart = new Vector();     //Almacena los encuentros de la jornada
    private Command cmdSelect;
    private Command cmdPreview;
    private BetProjectMIDlet midlet;
    TbApuestas tabApuesta;

    //Maneja las opciones de apuesta (pronóstico) del encuentro seleccionado
    public class TbApuestas extends SubList{
        BetProjectMIDlet midlet;
        Command cmdSave;
        String[] options;
        int selection;  //ID del encuentro, emp. desde cero
        String result;  //Pronóstico del encuentro, formato: ECU_ARG_110
        Alert alrConfDatos;     //Alert que indica que la info. ha sido almacenada
        Alert alrNoOptions;     //Maneja opciones no seleccionadas
        boolean addData;    //TRUE para añadir dato, FALSE para reemplazar dato
        boolean flagOptions;    //Controla que al menos una opción se haya seleccionada

        //Constructor
        public TbApuestas(String title, int type, BetProjectMIDlet midlet, Displayable
parent){
            super(title, type, midlet, parent);
            this.midlet = midlet;
            alrConfDatos = new Alert(null);
            alrNoOptions = new Alert(null);
            cmdSave = new Command("Guard.", Command.SCREEN, 1);   //Guardará los datos de
la apuesta realizada

            addCommand(cmdSave);
        }
        //Presenta las opciones del encuentro i-esimo para realizar pronóstico
        //y setea "addData" para añadir o reemplazar datos
        void setOptions(int selection, String[] options, boolean addData){
            this.options = options;
            this.selection = selection;
            this.addData = addData;

            deleteAll();
            append(" " + options[0], null);   //Equipo 1
            append(" " + options[1], null);   //Empate
            append(" " + options[2], null);   //Equipo 2
        }
        //Obtención de las opciones seleccionadas
        void dataforecast(){
```

```
            boolean[] selects = new boolean[size()];
            getSelectedFlags(selects);
            //Arma el dato que será almacenado en memoria, formato: ECU_ARG_110
            result =  options[0].substring(0, 3) + "_" +
                      options[2].substring(0, 3) + "_" +
                      (selects[0]? "1":"0")+
                      (selects[1]? "1":"0")+
                      (selects[2]? "1":"0");

            if(selects[0]||selects[1]||selects[2])
                flagOptions = true;
            else
                flagOptions = false;
        }

        public void commandAction(Command cmd, Displayable dis){
            super.commandAction(cmd, dis);
            if(cmd == cmdSave){
                dataforecast(); //Arma el dato que será almacenado en memoria
                if(flagOptions){
                    if(addData) //Añade dato
                        midlet.rsdatosEnc.addData(selection, result, true);
                    else    //Reemplaza dato
                        midlet.rsdatosEnc.addData(selection, result, false);

                    //Presenta Alert de almacenamiento de info.
                    Image imgConf = null;
                    try{
                        imgConf = Image.createImage("/save.png");
                    }catch(Exception e){
                        System.out.println("Except: TbEncuentros/TbApuestas/try del img: "
+ e.getMessage());
                    }
                    alrConfDatos.setImage(imgConf);
                    alrConfDatos.setString("Guardado en la memoria del teléfono.");
                    alrConfDatos.setTimeout(1200);
                    midlet.myDisplay.setCurrent(alrConfDatos, parent);
                }
                else{
                    //Presenta mensaje de error de no haber op. seleccionada
                    Image img = null;
                    try{
                        img = Image.createImage("/alert.png");
                    }catch(Exception e){
                        System.out.println("Except: TbEncuentros/TbApuestas/try del img: "
+ e.getMessage());
                    }
                    alrNoOptions.setImage(img);
                    alrNoOptions.setString("No se ha seleccionada ninguna opción. " +
                    "Favor seleccione alguna.");
                    alrNoOptions.setTimeout(1500);
                    midlet.myDisplay.setCurrent(alrNoOptions, parent);
                }
             }
        }
    }

    public TbEncuentros(String title, int type, BetProjectMIDlet midlet, Displayable
parent){
        super(title, type, midlet, parent);

        this.midlet = midlet;
        cmdSelect = new Command("Selec.", Command.SCREEN, 1);
        tabApuesta = new TbApuestas(null, MULTIPLE_LIST, midlet, this);
        Thread t = new Thread(this);    //Crea un thread para la conexión
        t.start();

        addCommand(cmdSelect);
    }

    public void run() {
        HttpConnection c = null;
        InputStream is = null;
        try{
            String url = midlet.getAppProperty("BaseURL") + "?queryStr=TabApuestas";
            //Estableciendo parametros para la conexion
            c = (HttpConnection)Connector.open(url);
```

```
            c.setRequestMethod(HttpConnection.GET);
            c.setRequestProperty("Content-Language", "es-ES");
            c.setRequestProperty("User-Agent", "Profile/MIDP-2.0 Configuration/CLDC-1.0");
            c.setRequestProperty("Connection", "Close");
            is = c.openInputStream();
            procesarRespuesta(c, is);
        }catch(Exception e){
            Alert alTCPError = new Alert(null);
            Image img = null;
            try{
                img = Image.createImage("/error2.png");
                alTCPError.setImage(img);
            }catch(Exception ex){
                System.out.println("Except: TbPositions/try img: " + ex.getMessage());
            }
            alTCPError.setString("No se pudo establecer comunicación con el proveedor" +
            " de la información. Favor, intente más tarde.");
            alTCPError.setTimeout(Alert.FOREVER);
            midlet.myDisplay.setCurrent(alTCPError, midlet.lsMainMenu);

            System.out.println("Except: TbEncuentros/run: " + e.getMessage());
        }finally {
             try{
                 if (is != null) is.close();
                 if (c != null) c.close();
             }catch(Exception e){
                 System.out.println("Except: Communication/run/finally/c: " +
e.getMessage());
             }
       }
    }

    public void procesarRespuesta(HttpConnection c, InputStream is)
    throws IOException{
        if(c.getResponseCode() == HttpConnection.HTTP_OK){
           int len = (int)c.getLength();
           //Procesamiento info con longitud conocida
           if(len > 0 ){
             int actual = 0;
             int bytesread = 0 ;
             byte[] data = new byte[len];

             while ((bytesread != len) && (actual != -1)) {
                actual = is.read(data, bytesread, len - bytesread);
                bytesread += actual;
                strResp = new String(data);
           }
             //Divide el stream (bytes) de datos en encuentros y equipos
             dividirDatos(data, len);
           }
            else{
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                int ch;
                while((ch = is.read()) != -1)
                    baos.write(ch);
                strResp = new String(baos.toByteArray());
                byte[] data1 = new byte[baos.size()];
                data1 = baos.toByteArray();
                //Divide el stream de datos (bytes) en encuentros y equipos
                dividirDatos(data1, len);
                baos.close();
            }
        }
        else{
            System.out.println("Except TbEncuentros/procesarR: "+c.getResponseMessage());
        }
    }

    public void dividirDatos(byte data[], int len){
        String strdata = "", strdata2 = ""; //Auxiliares
        byte[] byteDataEnc = new byte[30]; //Almacena los bytes de datos con los equipos de
c/encuentro
        int j=0;    //Contador auxiliar

        for(int i=0; i<len; i++){
            if(data[i] != 13){
                strdata = strdata + (char) data[i];
```

```
                byteDataEnc[j] = data [i];
                j++;
            }
            else{
                //Items de los encuentros
                datapart.addElement(strdata);
                strdata = "";
                i++;
                //Extracción de equipos de c/encuentro
                int k, aux;
                for(k=2; k<j; k++){
                    strdata2 = strdata2 + (char) byteDataEnc[k];
                    aux = k;
                    if(byteDataEnc[aux+1] == 32){
                        itemsEnc.addElement((String)strdata2);
                        strdata2 = "";  k+=3;   //Control de variables
                    }
                }
                itemsEnc.addElement((String)strdata2);
                strdata2 = "";  j=0;    //Control de variables
            }
        }
        //Añade los items de c/encuentro a la lista: Encuentros Jornada
        for(int i =0; i<datapart.size(); i++)
            append((String)datapart.elementAt(i), null);
    }

    public void commandAction(Command cmd, Displayable dis){
        super.commandAction(cmd, dis);
        if(cmd == cmdSelect || cmd == List.SELECT_COMMAND){
            int selection = getSelectedIndex();    //Almacena el # de encuentro
seleccionado, ini 0
            //Número de encuentros variable
            tabApuesta.setTitle("Pronóstico Enc. " + (selection+1));
            //Presentación de los items para el pronóstico
            //Previo se verifica si no hay ya un pronóstico de ese encuentro
            int numRecords = -1;
            try{
                numRecords = midlet.rsdatosEnc.rs.getNumRecords();
            }catch(Exception e){
                System.out.println("Except: TbEncuentros/commandAction/try: " +
e.getMessage());
            }
            //1era. ejecución, # registros = 0
            if(numRecords != -1&&numRecords == 0){
                //Divide a los equipos del encuentro para su presentación
                String[] optionsEnc = {(String)itemsEnc.elementAt(2*selection), "Empate",
(String)itemsEnc.elementAt(2*selection+1)};
                tabApuesta.setOptions(selection, optionsEnc, true);  //Presenta las
opciones en un List
                midlet.myDisplay.setCurrent(tabApuesta);    //Presenta las op. en pantalla
           }
            //Ya existen registros, # registros > 0
            else if( numRecords > 0){
                //Validar si el enc. seleccionado ya tiene un pronóstico almacenado
                boolean existDatos = midlet.rsdatosEnc.checkDataSelectd(selection);
                //EXISTE pronóstico previo
                if(existDatos){
                    boolean[] selectdOpts = new boolean[3];
                    selectdOpts = midlet.rsdatosEnc.retSelectdFlags();
                    //Seteo de las opciones almacenadas
                    String[] optionsEnc = {(String)itemsEnc.elementAt(2*selection),
"Empate", (String)itemsEnc.elementAt(2*selection+1)};
                    tabApuesta.setOptions(selection, optionsEnc, false);  //Presenta las
opciones en un List
                    tabApuesta.setSelectedFlags(selectdOpts);
                    midlet.myDisplay.setCurrent(tabApuesta);    //Presenta las op. en
pantalla
                }
                //NO EXISTE existe pronóstico previo
                else{
                    //Divide a los equipos del encuentro para su presentación
                    String[] optionsEnc = {(String)itemsEnc.elementAt(2*selection),
"Empate", (String)itemsEnc.elementAt(2*selection+1)};
                    tabApuesta.setOptions(selection, optionsEnc, true);  //Presenta las
opciones en un List
```

```
                    midlet.myDisplay.setCurrent(tabApuesta);    //Presenta las op. en
pantalla
                }
            }
        }
    }
}
```

Clase: TbPositions.java

Código Fuente:

```
/*
 * SubForm.java
 *
 * Created on 7 de febrero de 2005, 11:58 PM
 */

//package BetProjectMIDlet;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.*;
import java.util.Vector;


/**
 *
 * @author  Dalton Gordillo
 * @version
 */

//Implementa un Form donde se muestran las posiciones de los equipos
public class TbPositions extends SubForm implements Runnable{
    private String strResp = "";
    private StringItem stiData;
    private BetProjectMIDlet midlet;

    public TbPositions(String title, BetProjectMIDlet midlet, Displayable parent){
        super(title, midlet, parent);

        this.midlet = midlet;
        stiData = new StringItem(null, null);
        Thread t = new Thread(this);    //Crea un thread para la conexión
        t.start();

        append(stiData);
    }

    public void run() {
            HttpConnection c = null;
            InputStream is = null;
            try{
                String url = midlet.getAppProperty("BaseURL") + "?queryStr=TabPositions";
                c = (HttpConnection)Connector.open(url);
                c.setRequestMethod(HttpConnection.GET);
                c.setRequestProperty("Content-Language", "es-ES");
                c.setRequestProperty("User-Agent", "Profile/MIDP-2.0 Configuration/CLDC-
1.0");
                c.setRequestProperty("Connection", "Close");
                is = c.openInputStream();
                procesarRespuesta(c, is);
            }catch(Exception e){
                Alert alTCPError = new Alert(null);
                Image img = null;
                try{
                    img = Image.createImage("/error2.png");
                    alTCPError.setImage(img);
                }catch(Exception ex){
                    System.out.println("Except: TbPositions/try img: " + ex.getMessage());
                }
                alTCPError.setString("No se pudo establecer comunicación con el proveedor"
+
                " de la información. Favor, intente más tarde.");
                alTCPError.setTimeout(Alert.FOREVER);
                midlet.myDisplay.setCurrent(alTCPError, midlet.lsMainMenu);

                System.out.println("Except: TbPositions/run: " + e.getMessage());
            }finally {
                 try{
                     if (is != null) is.close();
```

```
                    if (c != null) c.close();
                }catch(Exception e){
                    System.out.println("Except: TbPositions/run/finally: " +
e.getMessage());
                }
             }
        }

        public void procesarRespuesta(HttpConnection c, InputStream is)
        throws IOException{
            if(c.getResponseCode() == HttpConnection.HTTP_OK){
               int len = (int)c.getLength();
               if(len > 0 ){
                 int actual = 0;
                 int bytesread = 0 ;
                 byte[] data = new byte[len];

                 while ((bytesread != len) && (actual != -1)) {
                    actual = is.read(data, bytesread, len - bytesread);
                    bytesread += actual;
                    strResp = new String(data);
                 }
               }
                else{
                    ByteArrayOutputStream baos = new ByteArrayOutputStream();
                    int ch;
                    while((ch = is.read()) != -1)
                        baos.write(ch);
                    strResp = new String(baos.toByteArray());
                    baos.close();
                }
                stiData.setText(strResp);
            }
            else{
                System.out.println("Error: "+c.getResponseMessage());
            }
        }
}
```

Clase: Base64Coder.java

Código Fuente:

```
/*
 * Base64Coder.java
 *
 * Created on 28 de abril de 2005, 09:58 PM
 */

//package BetProjectMIDlet;

/************************************************************************
*
* A Base64 Encoder/Decoder.
*
* This class is used to encode and decode data in Base64 format
* as described in RFC 1521.
*
* <p>
* Copyright 2003: Christian d'Heureuse, Inventec Informatik AG, Switzerland.<br>
* License: This is "Open Source" software and released under the <a
href="http://www.gnu.org/licenses/gpl.html" target="_top">GNU/GPL</a> license.
* It is provided "as is" without warranty of any kind. Please contact the author for other
licensing arrangements.<br>
* Home page: <a href="http://www.source-code.biz" target="_top">www.source-code.biz</a><br>
*
* <p>
* Version history:<br>
* 2003-07-22 Christian d'Heureuse (chdh): Module created.
*
************************************************************************/

public class Base64Coder {

// Mapping table from 6-bit nibbles to Base64 characters.
private static char[]    map1 = new char[64];
   static {
      int i=0;
      for (char c='A'; c<='Z'; c++) map1[i++] = c;
      for (char c='a'; c<='z'; c++) map1[i++] = c;
      for (char c='0'; c<='9'; c++) map1[i++] = c;
      map1[i++] = '+'; map1[i++] = '/'; }

// Mapping table from Base64 characters to 6-bit nibbles.
private static byte[]    map2 = new byte[128];
   static {
      for (int i=0; i<map2.length; i++) map2[i] = -1;
      for (int i=0; i<64; i++) map2[map1[i]] = (byte)i; }

/**
* Encodes a string into Base64 format.
* No blanks or line breaks are inserted.
* @param s  a String to be encoded.
* @return   A String with the Base64 encoded data.
*/
public static String encode (String s) {
   return new String(encode(s.getBytes())); }

/**
* Encodes a byte array into Base64 format.
* No blanks or line breaks are inserted.
* @param in  an array containing the data bytes to be encoded.
* @return    A character array with the Base64 encoded data.
*/
public static char[] encode (byte[] in) {
   int iLen = in.length;
   int oDataLen = (iLen*4+2)/3;       // output length without padding
   int oLen = ((iLen+2)/3)*4;         // output length including padding
   char[] out = new char[oLen];
   int ip = 0;
   int op = 0;
   while (ip < iLen) {
      int i0 = in[ip++] & 0xff;
```

```
      int i1 = ip < iLen ? in[ip++] & 0xff : 0;
      int i2 = ip < iLen ? in[ip++] & 0xff : 0;
      int o0 = i0 >>> 2;
      int o1 = ((i0 &   3) << 4) | (i1 >>> 4);
      int o2 = ((i1 & 0xf) << 2) | (i2 >>> 6);
      int o3 = i2 & 0x3F;
      out[op++] = map1[o0];
      out[op++] = map1[o1];
      out[op] = op < oDataLen ? map1[o2] : '='; op++;
      out[op] = op < oDataLen ? map1[o3] : '='; op++; }
   return out; }

/**
* Decodes a Base64 string.
* @param s  a Base64 String to be decoded.
* @return   A String containing the decoded data.
* @throws   IllegalArgumentException if the input is not valid Base64 encoded data.
*/
public static String decode (String s) {
   return new String(decode(s.toCharArray())); }

/**
* Decodes Base64 data.
* No blanks or line breaks are allowed within the Base64 encoded data.
* @param in  a character array containing the Base64 encoded data.
* @return    An array containing the decoded data bytes.
* @throws    IllegalArgumentException if the input is not valid Base64 encoded data.
*/
public static byte[] decode (char[] in) {
   int iLen = in.length;
   if (iLen%4 != 0) throw new IllegalArgumentException ("Length of Base64 encoded input
string is not a multiple of 4.");
   while (iLen > 0 && in[iLen-1] == '=') iLen--;
   int oLen = (iLen*3) / 4;
   byte[] out = new byte[oLen];
   int ip = 0;
   int op = 0;
   while (ip < iLen) {
      int i0 = in[ip++];
      int i1 = in[ip++];
      int i2 = ip < iLen ? in[ip++] : 'A';
      int i3 = ip < iLen ? in[ip++] : 'A';
      if (i0 > 127 || i1 > 127 || i2 > 127 || i3 > 127)
         throw new IllegalArgumentException ("Illegal character in Base64 encoded data.");
      int b0 = map2[i0];
      int b1 = map2[i1];
      int b2 = map2[i2];
      int b3 = map2[i3];
      if (b0 < 0 || b1 < 0 || b2 < 0 || b3 < 0)
         throw new IllegalArgumentException ("Illegal character in Base64 encoded data.");
      int o0 = ( b0       <<2) | (b1>>>4);
      int o1 = ((b1 & 0xf)<<4) | (b2>>>2);
      int o2 = ((b2 &   3)<<6) |  b3;
      out[op++] = (byte)o0;
      if (op<oLen) out[op++] = (byte)o1;
      if (op<oLen) out[op++] = (byte)o2; }
   return out; }

}
```

ANEXO No. 2: CÓDIGO FUENTE DEL SERVLET Y DE LA CLASE BASE64CODER

Servlet: BetProjectServlet.java

Código Fuente:

```
/*
 * BetProjectServlet.java
  * Created on 7 de marzo de 2005, 07:35 PM
 */

/**
 *
 * @author  Dalton Gordillo
 */

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.util.Properties.*;
import java.sql.*;
import java.lang.*;


public class BetProjectServlet extends HttpServlet{
    /** Creates a new instance of BetProjectServlet */
    int accessCont = 0;    //Verifica un psw para el user (1) y dato válido (2)
    int errCount = 0;   //Cuenta psw incorrectos, al 3ro. se bloquea
    private Vector datapartSp = new Vector();
    String user = null;
    int rowsChg = -1;

//Controla los datos de ususario (user y password)
    private void userValidation(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{
        accessCont = 0;
        //Obtiene del header de Authorization
        String authHeader = request.getHeader("Authorization");
        //Obtiene el psw en Base64
        String encodedData = authHeader.substring(6, authHeader.length());
        //Descodidificación de los datos y obtención del user
        String decodedData = Base64Coder.decode(encodedData);
        int length = decodedData.length();
        int indexSep = decodedData.indexOf(":");
        String userMob = null;     //User tx por móvil
        String pswMob = null;  //Psw tx por móvil
        //Obtiene los datos de user y psw decodificados
        userMob = decodedData.substring(0, indexSep);
        pswMob = decodedData.substring(indexSep+1, length);
        setUser(userMob);
        //Consulta del psw con el user
//**************************    Código para consultar a la DB
        String pswEncodedDB = null;
        Statement st = null;
        ResultSet rs = null;
        try{
            //Consulta de datos en la DB MYSQL
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/betproject");
            st = con.createStatement();
            if(st.execute("SELECT password FROM usersinfo WHERE username =""+userMob+"""")){
                rs = st.getResultSet();
                if(rs.next()){
                    pswEncodedDB = rs.getString("password");
                    accessCont = 1;
                }
            }
        } catch (Exception e) {
            System.out.println("Except: doPost/try query DB: " + e.getMessage());
        }finally{
            if(rs != null){
                try{
                    rs.close();
                }catch(Exception e){}
```

```
                rs = null;
            }
            if(st != null){
                try{
                    st.close();
                }catch(Exception e){}
                st = null;
        }
    }
//***************************
        //Decodifica el psw almacenado en la DB y compara con el enviado por el móvil.
        String pswDecodedDB = null;
        if(accessCont == 1){
            //Usuario registrado, se procede a validar password
            pswDecodedDB = Base64Coder.decode(pswEncodedDB);
            if(pswDecodedDB!=null && pswMob!=null){
                if(pswDecodedDB.equals(pswMob)){
                    //Psw OK, se fija accessCont y se encera errCount
                    accessCont = 2;
                    errCount = 0;
                    System.out.println("User data OK...");
                }
                else{
                    //Psw ERROR, se incrementa errCount y se envía mensaje de error
                    response.setHeader("WWW-Authenticate", "BASIC realm=badPassword");
                    response.sendError(response.SC_UNAUTHORIZED);
                    errCount++;
                    System.out.println("User data Invalid!!!");
                }
            }
        }
        else{
            //Usuario no registrado
            response.setHeader("WWW-Authenticate", "BASIC realm=noUserReg");
            response.sendError(response.SC_UNAUTHORIZED);
            System.out.println("No user Registered!!!");
            errCount++;
        }
    }

    public void splitData(String forecast){
        int len = forecast.length();
        int len2 = len/13;
        int j = 0;
        datapartSp.removeAllElements();
        int control = 0;

        for(int i=0; i<len2; i++){
            datapartSp.addElement(forecast.substring(j, j+12));
            j+=13;
            control++;
        }

        if(len2 != 5){
            for(int k = control; k<5; k++)
                datapartSp.addElement("NoData");
        }
    }

    public void setUser(String user){
        this.user = user;
    }

    public void loadDataToDB(){
        String pswEncodedDB = null;
        Statement st = null;
        ResultSet rs = null;
        try{
            //Consulta de datos en la DB MYSQL
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/betproject");
            st = con.createStatement();
            String sql = "INSERT INTO forecast_users VALUES("" + user + "","" +
            datapartSp.elementAt(0) + "","+
            datapartSp.elementAt(1) + "","" + datapartSp.elementAt(2) + "","" +
            datapartSp.elementAt(3) + "","" + datapartSp.elementAt(4) + "")";
```

```
            rowsChg = st.executeUpdate(sql);
        } catch (Exception e) {
            System.out.println("Except: doPost/try query DB: " + e.getMessage());
        }finally{
            if(rs != null){
                try{
                    rs.close();
                }catch(Exception e){}
                rs = null;
            }
            if(st != null){
                try{
                    st.close();
                }catch(Exception e){}
                st = null;
            }
        }
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{

        response.setContentType("text/plain");
        PrintWriter out = null;
        try{
            String query = request.getParameter("queryStr");

            if(query.compareTo("TabApuestas") == 0){
                out = response.getWriter();
//****************************    Código para consultar a la DB
                String strTotalEnc = "";
                Statement st = null;
                ResultSet rs = null;
                try{
                    //Consulta de datos en la DB MYSQL
                    Class.forName("com.mysql.jdbc.Driver");//.newInstance();
                    Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/betproject");
                    st = con.createStatement();
                    rs = st.executeQuery("SELECT * FROM encuentros");
                    String strItem = "";
                    String equipo_a = "";
                    String equipo_b = "";
                    while(rs.next()){

                        strItem = strItem.valueOf(rs.getInt("item")).trim();
                        equipo_a = rs.getString("equipo_a").trim();
                        equipo_b = rs.getString("equipo_b").trim();
                        strTotalEnc = strTotalEnc.concat(strItem + ":" + equipo_a + " - "
+equipo_b + "\r\n");
                    }
                    out.print(strTotalEnc);
                } catch (Exception e) {
                    System.out.println("Except: doPost/try query DB: " + e.getMessage());
                }finally{
                    if(rs != null){
                        try{
                            rs.close();
                        }catch(Exception e){}
                        rs = null;
                    }
                    if(st != null){
                        try{
                            st.close();
                        }catch(Exception e){}
                        st = null;
                    }
                }
//****************************
                }
            else if(query.compareTo("TabPositions") == 0){
                out = response.getWriter();
//****************************    Código para consultar a la DB
                String positions = "";
                Statement st = null;
                ResultSet rs = null;
                try{
```

```
                    //Consulta de datos en la DB MYSQL
                    Class.forName("com.mysql.jdbc.Driver");//.newInstance();
                    Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/betproject");
                    st = con.createStatement();
                    rs = st.executeQuery("SELECT * FROM tab_positions");
                    while(rs.next())
                        positions = positions + rs.getInt("item")+":
"+rs.getString("equipo")
                        +":Pts="+rs.getInt("puntos")+",Dif="+rs.getInt("g_dif")+"\r\n";

                    out.print(positions);
                    //System.out.println(positions);
                } catch (Exception e) {
                    System.out.println("Except: doPost/try query DB: " + e.getMessage());
                }finally{
                    if(rs != null){
                        try{
                            rs.close();
                        }catch(Exception e){}
                        rs = null;
                    }
                    if(st != null){
                        try{
                            st.close();
                        }catch(Exception e){}
                        st = null;
                    }
                }
//***************************
            }
            else{
                response.setContentLength("Tabla No Disponible".length());
                out = response.getWriter();
                out.println("Tabla No Disponible");
            }
            out.close();
        }catch(Exception e){
            System.out.println("Except: doGet/try: " + e.getMessage());
        }
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{

        response.setContentType("text/plain");
        PrintWriter out = null;
        try{
            String query = request.getParameter("queryStr");
            //Validación de los datos de usario
            if(query.compareTo("UserValidation") == 0){
                userValidation(request, response);
            }

            else if(query.compareTo("DataForescast") == 0){
                if(accessCont == 2){
                    //Recibe información del móvil para el pronóstico
                    BufferedReader reader = request.getReader();
                    char inData[] = new char[request.getIntHeader("Content-Length")];
                    reader.read(inData, 0, inData.length);
                    String totalForecast = new String(inData);
                    splitData(totalForecast);

                    //Sube la información a la DB MYSQL
                    loadDataToDB();
                    //Confirmación de la transacción
                    out = response.getWriter();
                    out.print(rowsChg);
                    out.close();
                }
            }

        }catch(Exception e){
            System.out.println("Except: doPost/try/UserValidation: " + e.getMessage());
        }
    }
}
```

Clase: Base64Coder.java

Código Fuente:

```
/*
 * Base64Coder.java
 *
 * Created on 28 de abril de 2005, 10:22 PM
 */

public class Base64Coder {

// Mapping table from 6-bit nibbles to Base64 characters.
private static char[]    map1 = new char[64];
   static {
      int i=0;
      for (char c="A"; c<="Z"; c++) map1[i++] = c;
      for (char c="a"; c<="z"; c++) map1[i++] = c;
      for (char c="0"; c<="9"; c++) map1[i++] = c;
      map1[i++] = "+"; map1[i++] = "/"; }

// Mapping table from Base64 characters to 6-bit nibbles.
private static byte[]    map2 = new byte[128];
   static {
      for (int i=0; i<map2.length; i++) map2[i] = -1;
      for (int i=0; i<64; i++) map2[map1[i]] = (byte)i; }

/**
* Encodes a string into Base64 format.
* No blanks or line breaks are inserted.
* @param s  a String to be encoded.
* @return   A String with the Base64 encoded data.
*/
public static String encode (String s) {
   return new String(encode(s.getBytes())); }

/**
* Encodes a byte array into Base64 format.
* No blanks or line breaks are inserted.
* @param in  an array containing the data bytes to be encoded.
* @return    A character array with the Base64 encoded data.
*/
public static char[] encode (byte[] in) {
   int iLen = in.length;
   int oDataLen = (iLen*4+2)/3;       // output length without padding
   int oLen = ((iLen+2)/3)*4;         // output length including padding
   char[] out = new char[oLen];
   int ip = 0;
   int op = 0;
   while (ip < iLen) {
      int i0 = in[ip++] & 0xff;
      int i1 = ip < iLen ? in[ip++] & 0xff : 0;
      int i2 = ip < iLen ? in[ip++] & 0xff : 0;
      int o0 = i0 >>> 2;
      int o1 = ((i0 &   3) << 4) | (i1 >>> 4);
      int o2 = ((i1 & 0xf) << 2) | (i2 >>> 6);
      int o3 = i2 & 0x3F;
      out[op++] = map1[o0];
      out[op++] = map1[o1];
      out[op] = op < oDataLen ? map1[o2] : "="; op++;
      out[op] = op < oDataLen ? map1[o3] : "="; op++; }
   return out; }

/**
* Decodes a Base64 string.
* @param s  a Base64 String to be decoded.
* @return   A String containing the decoded data.
* @throws   IllegalArgumentException if the input is not valid Base64 encoded data.
*/
public static String decode (String s) {
   return new String(decode(s.toCharArray())); }

/**
* Decodes Base64 data.
```

```
* No blanks or line breaks are allowed within the Base64 encoded data.
* @param in  a character array containing the Base64 encoded data.
* @return    An array containing the decoded data bytes.
* @throws    IllegalArgumentException if the input is not valid Base64 encoded data.
*/
public static byte[] decode (char[] in) {
   int iLen = in.length;
   if (iLen%4 != 0) throw new IllegalArgumentException ("Length of Base64 encoded input
string is not a multiple of 4.");
   while (iLen > 0 && in[iLen-1] == "=") iLen--;
   int oLen = (iLen*3) / 4;
   byte[] out = new byte[oLen];
   int ip = 0;
   int op = 0;
   while (ip < iLen) {
      int i0 = in[ip++];
      int i1 = in[ip++];
      int i2 = ip < iLen ? in[ip++] : "A";
      int i3 = ip < iLen ? in[ip++] : "A";
      if (i0 > 127 || i1 > 127 || i2 > 127 || i3 > 127)
         throw new IllegalArgumentException ("Illegal character in Base64 encoded data.");
      int b0 = map2[i0];
      int b1 = map2[i1];
      int b2 = map2[i2];
      int b3 = map2[i3];
      if (b0 < 0 || b1 < 0 || b2 < 0 || b3 < 0)
         throw new IllegalArgumentException ("Illegal character in Base64 encoded data.");
      int o0 = ( b0       <<2) | (b1>>>4);
      int o1 = ((b1 & 0xf)<<4) | (b2>>>2);
      int o2 = ((b2 &   3)<<6) |  b3;
      out[op++] = (byte)o0;
      if (op<oLen) out[op++] = (byte)o1;
      if (op<oLen) out[op++] = (byte)o2; }
   return out; }

}
```

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

AAA	Authorization, Authentication, and Accounting
AMS	Application Management System
API	Application Programming Interface. Especificación de como un programador que edita una aplicación, accede al comportamiento y al estado de las clases y objetos.
applet	Programa escrito en el lenguaje de programación Java, para ser ejecutado dentro de un web browser que sea compatible con la plataforma Java, como lo son HotJava o Netscape Navigator.
AWT	Abstract Window Toolkit. Colección de componentes GUI (Graphical User Interface) que fueron implementados utilizando la plataforma nativa de la versión de estos componentes.
BSC	Base Station Controller
BTS	Base Station Transceiver Subsystem
bytecode	Código de máquina independiente, generado por el compilador Java y ejecutado por el intérprete Java.
catch	Palabra clave utilizada en el lenguaje de programación Java, utilizada para declarar un bloque de sentencias, que serán ejecutadas cuando se produzca una "Java exception".
CDC	Connected Device Configuration
CDMA	Code Division Multiple Access
CISC	Complex Instruction Set Computer
CLDC	Connected Limited Device Configuration
constructor	Es un pseudo método que crea un objeto. Los constructores son métodos instancia que tienen el mismo nombre que su clase y son invocados utilizando la palabra clave "new".
exception	Evento producido durante la ejecución de un programa, que evita que el mismo siga ejecutándose normalmente, por lo general se trata de un error. Java soporta "exceptions" con las palabras clave: try, catch y throw.

extends	Se utiliza esta palabra clave, para que una clase pueda heredar o añadir las funcionalidades de una superclase.
field	Dato miembro de una clase. Un "field" no es estático.
final	Palabra clave de Java que sirve para definir una entidad solamente una vez, la misma que luego no podrá ser cambiada o derivada.
finally	Palabra clave de Java que ejecuta un bloque de sentencias, ya sea que se produzca o no una "Java Exception", o también un error en tiempo de ejecución ocurrido en un bloque definido previamente con la palabra clave "try".
garbage collection	La detección y liberación automática de memoria, que ya no está siendo utilizada y es realizada por el "Java runtime".
gateway	Nodo en una red que sirve como interfase para entrar en otra red.
GCF	Generic Connection Framework
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
implements	Palabra clave de Java opcionalmente incluida en la declaración de la clase, para especificar cualquier interfase implementada por esta.
import	Palabra clave de Java utilizada al inicio de un archivo fuente para especificar las clases o el paquete entero a ser referido más tarde, sin tener que incluir el nombre del paquete en la referencia.
instance	(instancia) El objeto de una clase en particular.
interface	Palabra clave de Java utilizada para definir una colección de definiciones de métodos y valores constantes, que luego podrá ser implementada por las clases que definan esta interfase con la palabra clave "implements".
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JAD Files (.jar)	Java Application Descriptor File. Este archivo lista el nombre del archivo, los nombres de cada MIDlet y los de sus clases dentro del suite, así como otra información adicional.
JAE	Java Application Environment

JAR Files (.jar)	Java Archive File. Un tipo de archivo utilizado para agregar varios archivos en uno solo.
JCP	Java Community Proccess
JDBC	Java Database Conectivity. La interfase JDBC provee de las APIs para el acceso, basado en SQL, a un amplio rango de bases de datos.
JDK	Java Development Kit software
JFC	Java Foundation Class
JRE	Java Runtime Environment
JVM	Java Virtual Machine
KVM	Kilobyte Virtual Machine
método	(method) Función definida en una clase.
midlet	Aplicación desarrollada en base al perfil MIDP y está diseñada para ser ejecutada en dispositivos móviles inalámbricos de comunicación.
MIDP	Mobile Information Device Profile
MIME	Multipurpose Internet Mail Extensions
MS	Mobile Station
MSC	Mobile Switching Center
MTA	Metropolitan Trade Area
MVC	Model View Controller
new	Palabra clave de Java utilizada para crear una instancia de una clase.
null	Palabra clave de Java utilizada para especificar un valor no definido para referencia de variables.
package	Grupo de "types". Los "package" son declarados con la palabra clave "package".
PDAs	Personal Digital Assistants
PDN	Packed Data Network
PDSN	Packet Data Serving Node
Profiles	Colección de Java APIs que complementan una o más Java 2 Platform Editions, añadiendo capacidades específicas.
RFC	Request For Comments

RISC Reduced Instruction Set Computer

RMI Remote Method Invocation. Conjunto de protocolos desarrollado por Sun, que hacen posible la comunicación remota entre objetos Java.

RMS Record Management System

RTT Radio Transmission Technology

servlet Es un programa que funciona en el lado del servidor. Proporciona una funcionalidad adicional a los servidores que utilizan la tecnología Java.

SQL Structured Query Language

static Palabra clave de Java utilizada para definir una variable, como la variable de una clase.

subclase (subclass) Es una clase que es derivada de otra clase en particular, pudiendo tener una o más clases de por medio.

superclase (superclass) Es una clase de la cual se derivan otras clases, pudiendo tener una o más clases de por medio.

this Palabra clave de Java que puede ser utilizada para representar una instancia de una clase en donde esta aparece. "this" es utilizada para acceder a las variables de la clase y a los métodos.

thread Es la unidad básica de ejecución de un programa. Un procesopuede tener varios “Threads” ejecutándose a la vez, desmpeñando un trabajo diferente. Cuando un Thread finaliza su trabajo, es suspendido y destruido.

throws Palabra clave de Java utilizada en la declaración de un método, para especificar las excepciones ("exceptions") que no serán manejadas dentro del método, sino que serán pasadas al siguiente nivel del programa.

try Palabra clave de Java que define un bloque de sentencias que pueden lanzar una excepción ("exception"). Si una excepción es lanzada, un bloque opcional "catch" puede manejar las excepciones que se han especificado dentro del bloque "try". Por otro lado, un bloque "finally", que es opcional, puede ser ejecutado luego de que se haya o no lanzado una excepción.

URI	Uniform Resource Identifier
URL	Uniform Resource Locator. Es un estándar para escribir un texto de refencia para una porción de texto en la WWW.
UTC	Unified Testing Criteria
UTI	Unified Testing Initiative
VM	Virtual Machine. Una especificación abstracta para un dispositivo de cómputo que puede ser implementado de diferentes maneras, en software o hardware.

FECHA DE ENTREGA DEL PROYECTO

El presente proyecto de grado fue entregado a la Facultad de Ingeniería Electrónica y reposa en la Escuela Politécnica del Ejército desde:

Sangolquí, a _____ de septiembre del 2005

SR. TCRN. EM. XAVIER MARTÍNEZ
DECANO DE LA FACULTAD DE INGENIERÍA ELECTRÓNICA

DR. JORGE CARVAJAL
SECRETARIO ACADÉMICO

DALTON GORDILLO LARCO
AUTOR

FECHA DE ENTREGA DEL PROYECTO

El presente proyecto de grado fue entregado a la Facultad de Ingeniería Electrónica y reposa en la Escuela Politécnica del Ejército desde:

Sangolquí, a _____ del 2005

_____

SR. TCRN. EM. XAVIER MARTÍNEZ
DECANO DE LA FACULTAD DE INGENIERÍA ELECTRÓNICA

_____

DR. JORGE CARVAJAL
SECRETARIO ACADÉMICO

_____

DALTON GORDILLO LARCO
AUTOR