Veni

Contenido para las presentaciones de marketing – ANEXO 3
v1.0 (anexo completo y revisado)

Historial de cambios:

- Gorka 24.08.2016: elaboración del anexo.

Anexo 3: Creando una BotBasic app

[Nota: escrito en primera persona intencionalmente.]

Crearemos una BotBasic app simple cuyo objetivo será agilizar y controlar el proceso de solicitud de insumos de oficina para todas las áreas de la empresa. Este proceso comúnmente confronta demoras en la cadena interna de suministros debido a tres razones:

1. Dentro de cada unidad organizacional, es lento (por lo iterativo) ponerse de acuerdo sobre cuándo realmente se necesita pedir algún insumo. Generalmente se trata de una negociación entre los empleados y el jefe de la unidad.
2. Típicamente es necesario acumular requerimientos y consolidarlos en pocos pedidos. Esto hace que algunos insumos que se requieren con prontitud deban esperar por el llenado de una "lista mínima".
3. Se trata de un proceso del back-office, no medular para las empresas, cuya operación normalmente no está tan bien soportada tecnológicamente y a nivel de procesos como los procesos medulares.

Hay aplicaciones, como los sistemas ERP, que incluyen estas facilidades, pero asumiremos que nuestra empresa no dispone de esta tecnología.

El objetivo de la empresa al crear esta BotBasic app es ganar eficiencia en el suministro de insumos, control en el proceso y mejorar el clima laboral al reducir las ocurrencias de ausencia de insumos en sus distintas unidades organizacionales.

1. Analizar el problema y efectuar el diseño funcional de su solución

El problema permite definir los parámetros y características del "modelo de negocio". El análisis del problema, que es el primer paso en todo proyecto de desarrollo de software, incluye para BotBasic el funcionamiento deseado de la solución del problema.

Los diferentes insumos estarán agrupados en categorías. Cada solicitud es efectuada por un empleado. La solicitud incluye un insumo (en cuya descripción va la unidad de medida), la cantidad, la razón de la solicitud, algunas observaciones y el plazo en que se requiere (3 días / 7 días / 15 días / 1 mes / 3 meses).

El jefe de unidad valida y aprueba (o rechaza) las solicitudes. Las solicitudes aprobadas se

derivan directamente a la unidad de aprovisionamiento.

A cada solicitud se le asigna un código automáticamente, que puede ser revisado por el solicitante, junto con el estado de la solicitud, que puede ser:

1. Solicitado por el empleado.
2. Aprobado por el jefe de unidad.
3. Negado por el jefe de unidad.
4. Despachado por la unidad de aprovisionamiento.
5. Recibido por la unidad solicitante.

La unidad de aprovisionamiento construye "lotes", manualmente, a partir de los requerimientos y su dinámica de trabajo. Cada lote es alimentado en la BotBasic app y actúa como un contenedor de múltiples requerimientos realizados por la misma unidad.

Quienes pueden recibir un pedido son el empleado solicitante (un requerimiento en específico), el jefe de la unidad solicitante, y algunos empleados de la unidad que sean designados como "receptores de insumos". Al momento de recibir los insumos solicitados, la idea es que el jefe de unidad sólo deba recibir un lote (ya preparado por la unidad de aprovisionamiento) y no insumo por insumo, a fin de evitar consumo de su tiempo.

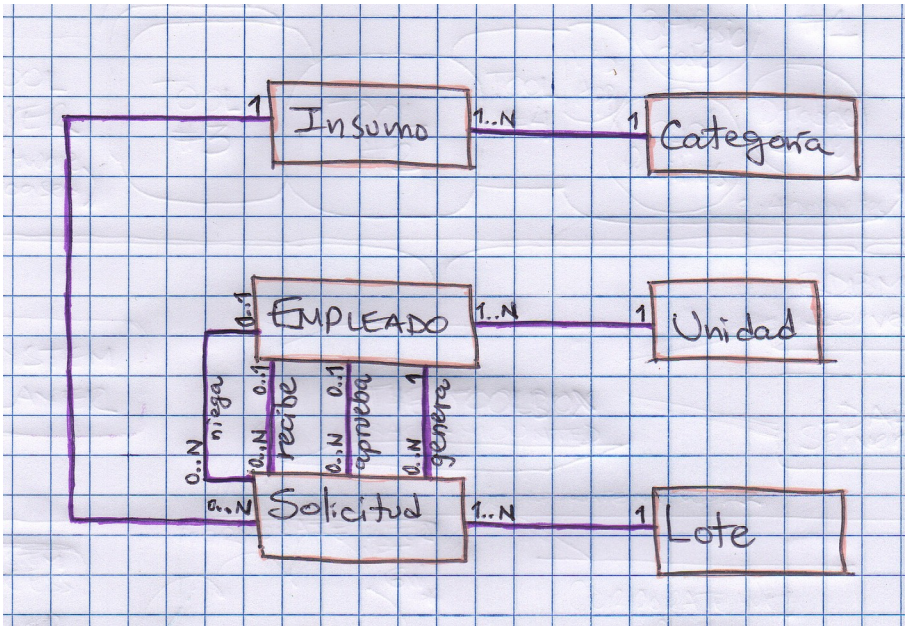
2. Crear el modelo de datos

El segundo paso se trata de diseñar el modelo entidad-relación y los atributos de las tablas SQL. Esto lo hace el programador de BotBasic (su perfil es conocer de bases de datos, PHP y BotBasic).

Por medio de una lectura de la formulación del análisis, se observa que las entidades involucradas y sus atributos son:

1. Insumo: nombre, categoría a la que pertenece.
2. Categoría (de insumo): nombre.
3. Solicitud: insumo, cantidad, código, fecha, plazo, observaciones, empleados involucrados según rol (genera, aprueba o niega, recibe), estado.
4. Empleado: unidad organizacional a la que pertenece, atributos que indican si puede aprobar solicitudes (como jefe de unidad) y si puede recibir insumos que llegan a la unidad.
5. Unidad (organizacional): nombre.
6. Lote: conjunto de solicitudes incluidas.

El modelo entidad relación (se muestra simplificado, pues no está el detalle de los atributos) tiene, entonces, 6 entidades y 8 relaciones:



Un modelo entidad-relación (E-R) es fácil de leer. Las entidades son las cajas con líneas naranja, mientras que las relaciones entre entidades son líneas violeta. Cada relación incluye descripción de multiplicidades y opcionalmente de roles:

1. Todo Insumo está asociado a exactamente una Categoría. Y cada Categoría está asociada a múltiples (y al menos un) Insumo. Es necesario distinguir en que extremo de la línea de relación se escribe el indicador de multiplicidad para cada caso.
2. Cada insumo está asociado a múltiples solicitudes (0..N: hay insumos que podrían no estar incluidos en solicitudes en un momento dado). Cada solicitud está asociada exactamente a un insumo.
3. Una Solicitud está asociada a exactamente un Lote. Cada Lote está asociado (incluye) a una o más solicitudes.
4. Cada empleado está asociado a exactamente una Unidad. Toda Unidad incluye al menos un empleado.
5. Cada Solicitud es generada por exactamente un Empleado (rol "genera"). Cada Empleado puede haber generado cualquier número de Solicitudes.
6. Cada Solicitud es aprobada exactamente por un Empleado, pero cuando la Solicitud no ha sido aprobada aún, la relación entre ambas estidades no ha sido construida. De allí la multiplicidad 0..1 en el extremo conectado a la caja Empleado.
7. Cada (resultado de la gestión por parte de la unidad de aprovisionamiento de una) Solicitud es recibida por exactamente un Empleado (cuando no ha habido entrega aún de los insumos, la relación no está construida). En este sistema, recibir un Lote se representará como la recepción de todas las Solicitudes incluidas en él.

Por simplicidad para este ejemplo, no se contemplan conexiones al servidor de directorio de la empresa, si hubiese (LDAP o equivalente). Es por eso que el sistema maneja las entidades propias Empleado y Unidad. La validación de usuarios se hace por medio del contraste de su número de teléfono móvil con los datos almacenados en la tabla Empleado. Cuando se implementan soluciones en las empresas, la BotBasic app incluye, como parte de la lógica de negocios, un procedimiento de validación del número de teléfono celular del empleado por medio

del envío de un PIN a través de un mensaje SMS o un e-mail.

3. Crear las tablas SQL

El tercer paso es construir las tablas del manejador de bases de datos relacionales (RDBMS). Estas tablas se construyen en lenguaje SQL por medio de scripts de creación, que son ejecutados en el RDBMS para ponerlas a disposición del software que está siendo creado.

Cada entidad del modelo E-R se representa por medio de una tabla. Cada tabla contiene los atributos de la entidad como columnas y los diferentes datos como filas, formando un reticulado parecido a una hoja de cálculo.

Las relaciones se expresan también como atributos, gracias la inclusión en cada tabla de un atributo "id" de tipo numérico y de generación por secuencia automática (de esto último se encarga el RDBMS). Una relación con multiplicidad singular (0..1 o 1) por un extremo y por el otro plural (0..N o 1..N) se traslada a SQL incluyendo el id de la entidad contraparte en la tabla de la entidad cuyo extremo de la relación aparece en plural. Por ejemplo, en la relación Solicitud-Lote, se incluye el id del Lote en la tabla Solicitud.

Este es el script de creación de las tablas del modelo E-R en SQL:

```
CREATE DATABASE IF NOT EXISTS aprovisionamiento;
USE aprovisionamiento;

CREATE TABLE categoria (
  id int(11) NOT NULL AUTO_INCREMENT,
  nombre varchar(64) NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE empleado (
  id int(11) NOT NULL AUTO_INCREMENT,
  nombre varchar(64) NOT NULL,
  celular VARCHAR(16) NOT NULL,
  puede_aprobar tinyint(1) NOT NULL,
  puede_recibir tinyint(1) NOT NULL,
  id_unidad int(11) NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE insumo (
  id int(11) NOT NULL AUTO_INCREMENT,
  nombre varchar(128) NOT NULL,
  id_categoria int(11) NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE lote (
  id int(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (id)
);

CREATE TABLE solicitud (
  id int(11) NOT NULL AUTO_INCREMENT,
  id_insumo int(11) NOT NULL,
  cantidad int(11) NOT NULL,
  fecha datetime NOT NULL,
  plazo enum('3 dias','7 dias','15 dias','1 mes','3 meses') NOT NULL,
  observaciones varchar(255),
```

```
  id_generador int(11) NOT NULL,
  id_aprobador int(11),
  id_negador int(11),
  id_recibidor int(11),
  id_lote int(11),
  estado enum('solicitado','aprobado','negado','despachado','recibido') NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE unidad (
  id int(11) NOT NULL AUTO_INCREMENT,
  nombre varchar(64) NOT NULL,
  PRIMARY KEY (id)
);
```

Por facilidad, en este ejemplo, no se incluyen restricciones referenciales en el código SQL mostrado. Así mismo, no se manejará el concepto de transacciones en el siguiente paso.

Ahora llenaremos las tablas Insumo, Categoría, Unidad y Empleado con datos de prueba, por medio del siguiente código en SQL:

```
INSERT INTO categoria VALUES(1, 'Papelería');
INSERT INTO categoria VALUES(2, 'Aseo');
INSERT INTO categoria VALUES(3, 'Alimentos');
INSERT INTO categoria VALUES(4, 'Informática');

INSERT INTO empleado VALUES(1, 'José Pérez', '99887701', 1, 0, 1);
INSERT INTO empleado VALUES(2, 'Juan Navarro', '99887702', 0, 0, 1);
INSERT INTO empleado VALUES(3, 'Ella Fitzgerald', '99887703', 0, 1, 1);
INSERT INTO empleado VALUES(4, 'Lana Ramírez', '99887704', 0, 0, 1);
INSERT INTO empleado VALUES(5, 'Miguel Zúñiga', '99887705', 1, 0, 2);
INSERT INTO empleado VALUES(6, 'María Reyes', '99887706', 1, 0, 3);
INSERT INTO empleado VALUES(7, 'John Esquivel', '99887707', 1, 0, 4);
INSERT INTO empleado VALUES(8, 'Marta De San Juan', '99887708', 0, 1, 4);

INSERT INTO insumo VALUES(1, 'Resma de papel carta 500 hojas', 1);
INSERT INTO insumo VALUES(2, 'Caja de 12 lápices #2', 1);
INSERT INTO insumo VALUES(3, 'Caja de 12 bolígrafos color negro', 1);
INSERT INTO insumo VALUES(4, 'Caja de 12 bolígrafos color azul', 1);
INSERT INTO insumo VALUES(5, 'Caja de 12 bolígrafos color rojo', 1);
INSERT INTO insumo VALUES(6, 'Rollo de cinta adhesiva con dispensador', 1);
INSERT INTO insumo VALUES(7, 'Caja de grapas corrugadas', 1);
INSERT INTO insumo VALUES(8, 'Caja de 100 clips', 1);
INSERT INTO insumo VALUES(9, 'Engrapadora para grapas corrugadas', 1);
INSERT INTO insumo VALUES(10, 'Pack de 5 marcadores para pizarra acrílica colores surtidos', 1);
INSERT INTO insumo VALUES(11, 'Jabón líquido de manos 300 cc', 2);
INSERT INTO insumo VALUES(12, 'Pack de 12 rollos de papel higiénico', 2);
INSERT INTO insumo VALUES(13, 'Detergente para inodoros 1 litro', 2);
INSERT INTO insumo VALUES(14, 'Paquete de 300 servilletas tipo restaurant', 2);
INSERT INTO insumo VALUES(15, 'Pack de 3 paños absorbentes de cocina', 2);
INSERT INTO insumo VALUES(16, 'Paquete de café molido 1 kg', 3);
INSERT INTO insumo VALUES(17, 'Caja de 30 bolsas de té negro', 3);
INSERT INTO insumo VALUES(18, 'Caja de 30 bolsas de té verde', 3);
INSERT INTO insumo VALUES(19, 'Paquete de azúcar molida 1 kg', 3);
INSERT INTO insumo VALUES(20, 'Botellón de agua 20 lts', 3);
INSERT INTO insumo VALUES(21, 'Toner para impresora láser departamental negro', 4);
INSERT INTO insumo VALUES(22, 'Toner para impresora láser departamental cyan', 4);
INSERT INTO insumo VALUES(23, 'Toner para impresora láser departamental magenta', 4);
INSERT INTO insumo VALUES(24, 'Toner para impresora láser departamental yellow', 4);
INSERT INTO insumo VALUES(25, 'Cartucho de tinta para impresora personal monocromática negro', 4);
INSERT INTO insumo VALUES(26, 'Cartucho de tinta para impresora personal a color negro', 4);
INSERT INTO insumo VALUES(27, 'Cartucho de tinta para impresora personal a color CMY', 4);
INSERT INTO insumo VALUES(28, 'Pen drive USB 8 GB', 4);
INSERT INTO insumo VALUES(29, 'Pen drive USB 32 GB', 4);
```

```
INSERT INTO unidad VALUES(1, 'Departamento de Operaciones Comerciales');
INSERT INTO unidad VALUES(2, 'Departamento de Operaciones Logísticas');
INSERT INTO unidad VALUES(3, 'Departamento de Soporte e Infraestructura');
INSERT INTO unidad VALUES(4, 'Gerencia de la Unidad Estratégica de Negocios');
```

El contenido de las tablas con los datos de prueba es:

Insumo:

id	nombre	id_categoria
1	Resma de papel carta 500 hojas	1
2	Caja de 12 lápices #2	1
3	Caja de 12 bolígrafos color negro	1
4	Caja de 12 bolígrafos color azul	1
5	Caja de 12 bolígrafos color rojo	1
6	Rollo de cinta adhesiva con dispensador	1
7	Caja de grapas corrugadas	1
8	Caja de 100 clips	1
9	Engrapadora para grapas corrugadas	1
10	Pack de 5 marcadores para pizarra acrílica colores surtidos	1
11	Jabón líquido de manos 300 cc	2
12	Pack de 12 rollos de papel higiénico	2
13	Detergente para inodoros 1 litro	2
14	Paquete de 300 servilletas tipo restaurant	2
15	Pack de 3 paños absorbentes de cocina	2
16	Paquete de café molido 1 kg	3
17	Caja de 30 bolsas de té negro	3
18	Caja de 30 bolsas de té verde	3
19	Paquete de azúcar molida 1 kg	3
20	Botellón de agua 20 lts	3
21	Toner para impresora láser departamental negro	4
22	Toner para impresora láser departamental cyan	4
23	Toner para impresora láser departamental magenta	4
24	Toner para impresora láser departamental yellow	4
25	Cartucho de tinta para impresora personal monocromática negro	4
26	Cartucho de tinta para impresora personal a color negro	4
27	Cartucho de tinta para impresora personal a color CMY	4
28	Pen drive USB 8 GB	4
29	Pen drive USB 32 GB	4

Categoría:

id	nombre
1	Papelería
2	Aseo
3	Alimentos
4	Informática

Unidad:

id	nombre
1	Departamento de Operaciones Comerciales
2	Departamento de Operaciones Logísticas
3	Departamento de Soporte e Infraestructura
4	Gerencia de la Unidad Estratégica de Negocios

Empleado:

id	nombre	celular	puede_aprobar	puede_recibir	id_unidad
1	José Pérez	99887701	1	0	1
2	Juan Navarro	99887702	0	0	1
3	Ella Fitzgerald	99887703	0	1	1
4	Lana Ramírez	99887704	0	0	1
5	Miguel Zúñiga	99887705	1	0	2
6	María Reyes	99887706	1	0	3
7	John Esquivel	99887707	1	0	4
8	Marta De San Juan	99887708	0	1	4

4. Crear las clases en PHP que acceden a los datos almacenados en las tablas

Normalmente, sobre las filas (registros) de las tablas se aplican operaciones tipo CRUD (create, read, update y delete). No siempre es necesario implementar todas las operaciones. Nuestra BotBasic app, por ejemplo, no requiere borrado de registros, pero sí creación, actualización y lectura.

El cuarto paso consiste en crear "clases" en lenguaje PHP que permitan efectuar operaciones sobre la información almacenada en la base de datos. Las clases son una forma ofrecida por los lenguajes que disponen de orientación a objetos para encapsular operaciones relacionadas.

Diseñaremos una clase para cada entidad y en cada una definiremos las operaciones necesarias, implementándolas por medio de código SQL.

Pero antes generaremos una clase PHP auxiliar que nos permitirá canalizar todas las operaciones de la base de datos; la llamaremos DBbroker:

```php
<?php

// use \PDO, \PDOException;
// esta clase usa a la clase Log.php, que maneja la bitacora de mensajes y errores

define('TTB_DB_DRIVER',   'mysql'    );   // uno de los posibles RDBMS a los que se accede via PHP Data Objects (PDO)
define('TTB_DB_HOST',     'localhost');
define('TTB_DB_NAME',     'ttbdb'    );
define('TTB_DB_USER',     'ttbuser'  );
define('TTB_DB_PASSWORD', 'ttbpass'  );
define('TTB_DB_CONN_STR', TTB_DB_DRIVER . ":host=" . TTB_DB_HOST . ";dbname=" . TTB_DB_NAME);

class DBbroker
{

    /** @var PDO */
    static private $dbh = null;

    static private function connect ($disconnectFirst = false)
    {
        if (! $disconnectFirst && self::$dbh !== null) { return true; }
        try {
            self::disconnect();
            self::$dbh = new PDO(TTB_DB_CONN_STR, TTB_DB_USER, TTB_DB_PASSWORD, [ PDO::ATTR_PERSISTENT => true ]);
            return true;
        } catch (PDOException $e) {
            Log::register(Log::TYPE_DATABASE, "No es posible conectarse a la BD de BotBasic",
                          [ Log::ATTRIB_EXCEPTION, $e ]);
            return false;
        }
    }

    static public function disconnect ()
    {
        self::$dbh = null;
    }

```

```php
    static private function dbError ()
    {
        if (self::$dbh === null) { return "DB_NOT_CONNECTED"; }
        $ei  = self::$dbh->errorInfo();
        $res = $ei[0] . ($ei[1] === null ? '' : '/' . $ei[1]) . ($ei[2] === null ? '/NONE' : '/' . $ei[2]);
        return $res;
    }

    static public function query ($sql, $when = null)
    {
        if (! self::connect() === null) {
            Log::register(Log::TYPE_DATABASE, "base de datos no conectada al intentar Query sobre: $sql", null);
            return [];
        }
        $res = self::$dbh->query($sql);
        if ($res === false) {
            $error = self::dbError() . ($when === null ? '' : " WHEN $when");
            Log::register(Log::TYPE_DATABASE, $error, null);
        }
        return $res;
    }

    static public function exec ($sql, $when = null, $returnLastInsertId = false)
    {
        if (self::connect() === null) {
            Log::register(Log::TYPE_DATABASE, "base de datos no conectada al intentar Exec sobre: $sql", null);
            return $returnLastInsertId ? -1 : 0;
        }
        $res = self::$dbh->exec($sql);
        if ($res === false) {
            $error = self::dbError() . ($when === null ? '' : " WHEN $when");
            Log::register(Log::TYPE_DATABASE, $error, null);
        }
        if ($returnLastInsertId) { return intval(self::$dbh->lastInsertId()); }
        else                     { return $res;                               }
    }

}

```

Empezaremos ahora con las operaciones más importantes asociadas a la entidad Lote:

1. Crear () --> id_lote
   Crea un nuevo lote y devuelte su id numérico recién creado por el RDBMS.
2. Solicitudes (id_lote) --> ids_solicitudes[]
   Recupera todos los id de las solicitudes relacionadas (incluidas) en el lote, o un valor vacío ("empty") si el lote no tiene solicitudes o no ha sido creado previamente. (*)
   La notación ids_solicitudes[] representa un arreglo ("[]") de distintos id_solicitud.
3. Recibir (id_lote, id_empleado)
   Genera en la base de datos todas las relaciones Empleado-Solicitud de rol "recibe".
   Para ello recupera todas las solicitudes del lote e invoca a la operación "recibir" de la clase Solicitud.
   Podemos notar que no se trata de una operación que ejecutará directamente código SQL, sino que invocará a una operación definida en otra clase. Así habrá otros casos.

(*) Cuando una operación resulte en una condición de error de tipo funcional, retornará "empty", que a nivel de código de BotBasic se representará como una cadena de texto vacía.

El siguiente código en PHP define las operaciones de la clase Lote:

```php
<?php

class Lote
{

    static public function crear ()
    {
        $sql = <<<END
            INSERT INTO lote;
END;
        return DBbroker::exec($sql, null, true);
    }

    static public function solicitudes ($idLote)
    {
        $sql = <<<END
            SELECT id_solicitud
              FROM solicitud
             WHERE id_lote = $idLote;
END;
        $rows = DBbroker::query($sql);
        $res = [];
        foreach ($rows as $row) { $res[] = $row["id_solicitud"]; }
        return $res;
    }

    static public function recibir ($idLote, $idEmpleado)
    {
        $solicitudes = self::solicitudes($idLote);
        foreach ($solicitudes as $solicitud) {
            Solicitud::recibir($solicitud, $idEmpleado);
        }
    }

    static public function esValido ($idLote)
    {
        $sql = <<<END
            SELECT COUNT(*) AS cuenta
              FROM lote
             WHERE id = $idLote;
END;
        $rows = DBbroker::query($sql);
        return $rows[0]["cuenta"] != 0;
    }

}
```

Estas son las operaciones más importantes asociadas a la entidad Solicitud:

1. Generar (id_empleado, id_insumo, cantidad, plazo, observaciones) --> id_solicitud
   Crea un nuevo registro de solicitud y devuelve su id recién creado por el RDBMS. Genera en la base de datos la relación Empleado-Solicitud de rol "genera".
2. Aprobar (id_solicitud, id_empleado)
   Genera en la base de datos la relación Empleado-Solicitud de rol "aprueba".
3. Negar (id_solicitud, id_empleado)
   Genera en la base de datos la relación Empleado-Solicitud de rol "niega".
4. Despachar (id_solicitud)
   Asigna el estado "despachado por la unidad de aprovisionamiento" a la solicitud.
5. Recibir (id_solicitud, id_empleado)
   Genera en la base de datos la relación Empleado-Solicitud de rol "recibe".
6. IncluirEnLote (id_solicitud, id_lote)
   Genera en la base de datos una relación Solicitud-Lote.
7. EnRango (dias_minimos, dias_maximos) --> id_solicitud[]
   Retorna un arreglo de id de solicitudes que deban ser entregados en la unidad solicitante

dentro del rango de días especificados, contados a partir del día actual. También permite identificar solicitudes con fecha de entrega vencida.

8. Datos (id_solicitud) --> nombre_insumo, nombre_categoria, cantidad, estado, fecha, plazo, observaciones, nombre_solicitante, nombre_aprobador, nombre_negador, nombre_recibidor, nombre_unidad
   Devuelve la siguiente información asociada a la solicitud: nombre del insumo, nombre de la categoría del insumo, cantidad de insumo que fue solicitado, estado de la solicitud, fecha de la solicitud, plazo máximo solicitado para su entrega por parte de la unidad de aprovisionamiento, observaciones, nombre del empleado que solicitó, nombre del empleado (jefe de unidad) que aprobó, nombre del empleado (jefe de unidad) que negó, nombre del empleado que recibió, nombre de la unidad organizacional asociada a los empleados.

El siguiente código en PHP define las operaciones de la clase Solicitud:

```php
<?php

class Solicitud
{

    static public function generar ($idEmpleado, $idInsumo, $cantidad, $plazo, $observaciones)
    {
        $sql = <<<END
            INSERT INTO solicitud (id_insumo, cantidad, fecha, plazo, observaciones, id_generador, estado)
            VALUES ($idInsumo, $cantidad, NOW(), '$plazo', '$observaciones', $idEmpleado, 'solicitado');
END;
        return DBbroker::exec($sql, null, true);
    }

    static public function aprobar ($idSolicitud, $idEmpleado)
    {
        $sql = <<<END
            UPDATE solicitud
              SET id_aprobador = $idEmpleado, estado = 'aprobado'
            WHERE id_solicitud = $idSolicitud;
END;
        DBbroker::exec($sql);
    }

    static public function negar ($idSolicitud, $idEmpleado)
    {
        $sql = <<<END
            UPDATE solicitud
              SET id_negador = $idEmpleado, estado = 'negado'
            WHERE id_solicitud = $idSolicitud;
END;
        DBbroker::exec($sql);
    }

    static public function despachar ($idSolicitud)
    {
        $sql = <<<END
            UPDATE solicitud
              SET estado = 'despachado'
            WHERE id_solicitud = $idSolicitud;
END;
        DBbroker::exec($sql);
    }
```

```
    static public function recibir ($idSolicitud, $idEmpleado)
    {
        $sql = <<<END
            UPDATE solicitud
               SET id_recibidor = $idEmpleado, estado = 'recibido'
             WHERE id_solicitud = $idSolicitud;
END;
        DBbroker::exec($sql);
    }

    static public function incluirEnLote ($idSolicitud, $idLote)
    {
        $sql = <<<END
            UPDATE solicitud
               SET id_lote = $idLote
             WHERE id_solicitud = $idSolicitud;
END;
        DBbroker::exec($sql);
    }

    static public function enRango ($diasMinimos, $diasMaximos)
    {
        $sql = <<<END
            SELECT id_solicitud
              FROM solicitud
             WHERE DATE_DIFF(NOW(), DATE_ADD(fecha, INTERVAL CASE WHEN plazo = '3 dias' THEN 3
                                                                  WHEN plazo = '7 dias' THEN 7
                                                                  WHEN plazo = '15 dias' THEN 15
                                                                  WHEN plazo = '1 mes' THEN 30
                                                                  WHEN plazo = '3 meses' THEN 90 END DAY)
                   BETWEEN $diasMinimos AND $diasMaximos;
END;
        $rows = DBbroker::query($sql);
        $res = [];
        foreach ($rows as $row) { $res[] = $row["id_solicitud"]; }
        return $res;
    }

    static public function datos ($idSolicitud)
    {
        $sql = <<<END
            SELECT i.nombre, c.nombre, s.cantidad, s.estado, s.fecha, s.plazo,
                   s.observaciones, es.nombre, ea.nombre, en.nombre, er.nombre, u.nombre
              FROM solicitud AS s
              JOIN insumo AS i ON s.id_insumo = i.id
              JOIN categoria AS c ON i.id_categoria = c.id
              JOIN empleado AS es ON s.id_solicitante = es.id
              JOIN unidad AS u ON es.id_unidad = u.id
              LEFT JOIN empleado AS ea ON s.id_aprobador = ea.id
              LEFT JOIN empleado AS en ON s.id_negador = en.id
              LEFT JOIN empleado AS er ON s.id_recibidor = er.id
             WHERE s.id = $idSolicitud;
END;
        $rows = DBbroker::query($sql);
        return count($rows) > 0 ? array_values($rows[1]) : array_fill(0, 12, '');
    }

    static public function plazos ()
    {
        return [ '3 dias', '7 dias', '15 dias', '1 mes', '3 meses' ];
    }

    static public function estado ($idSolicitud)
    {
        $datos = self::datos($idSolicitud);
        return $datos[3];
    }

}
```

Definiremos también operaciones auxiliares sobre las entidades Empleado, Categoría e Insumo:

Para Empleado:

```php
<?php

class Empleado
{

    static public function esValido ($celular)
    {
        $sql = <<<END
            SELECT id
              FROM empleado
             WHERE celular = '$celular';
END;
        $rows = DBbroker::query($sql);
        return count($rows) == 0 ? -1 : $rows[0]["id"];
    }

    static public function puedeAprobarNegar ($idEmpleado)
    {
        $sql = <<<END
            SELECT puede_aprobar
              FROM empleado
             WHERE id = '$idEmpleado';
END;
        $rows = DBbroker::query($sql);
        return count($rows) == 0 ? 0 : $rows[0]["puede_aprobar"] != 0;
    }

    static public function puedeRecibir ($idEmpleado)
    {
        $sql = <<<END
            SELECT puede_recibir
              FROM empleado
             WHERE id = '$idEmpleado';
END;
        $rows = DBbroker::query($sql);
        return count($rows) == 0 ? 0 : $rows[0]["puede_recibir"] != 0;
    }

```

```php
    static public function aprobador ($idEmpleado)
    {
        $sql = <<<END
            SELECT s.id AS id
              FROM empleado AS s
              JOIN empleado AS e ON e.id_unidad = s.id_unidad
             WHERE e.id = '$idEmpleado'
             LIMIT 1;
END;
        $rows = DBbroker::query($sql);
        return count($rows) == 0 ? 0 : $rows[0]["id"];
    }

    static public function puedeRecibirLote ($idEmpleado, $idLote)
    {
        // hallar la unidad del empleado recibidor
        $sql = <<<END
            SELECT id_unidad
              FROM empleado
             WHERE id = $idEmpleado;
END;
        $rows = DBbroker::query($sql);
        if (count($rows) == 0) { return false; }
        $idUnidadEmpleado = $rows["id_unidad"];
        // contrastar con la unidad de cada empleado solicitante de cada solicitud del lote
        $sql = <<<END
            SELECT e.id_unidad AS id_unidad
              FROM empleado AS e
              JOIN solicitud AS s ON s.id_generador = e.id
             WHERE s.id_lote = $idLote;
END;
        $rows = DBbroker::query($sql);
        foreach ($rows as $row) {
            if ($row["id_unidad"] != $idUnidadEmpleado) { return false; }
        }
        return true;
    }

}
```

Para Categoria:

```php
<?php

class Categoria
{

    static public function listar ()
    {
        $sql = <<<END
            SELECT id, nombre
              FROM categoria
             ORDER BY nombre ASC;
END;
        $rows = DBbroker::query($sql);
        $res = [];
        foreach ($rows as $row) { $res[] = $row['nombre'] . '|' . $row['id']; }
        return $res;
    }

}
```

Para Insumo:

```php
<?php

class Insumo
{

    static public function listar ($idCategoria)
    {
        $sql = <<<END
            SELECT id, nombre
              FROM insumo
             WHERE id_categoria = $idCategoria
             ORDER BY nombre ASC;
END;
        $rows = DBbroker::query($sql);
        $res = [];
        foreach ($rows as $row) { $res[] = $row['nombre'] . '|' . $row['id']; }
        return $res;
    }

}
```

La incorporación de nuevos registros en alguna las tablas maestras (Unidad, Empleado, Categoria e Insumo) puede ser gestionada por medio de una herramienta CRUD genérica que puede ser incluida como web application de back-office administrativo para un usuario del área de aprovisionamiento. (*)

Por simplicidad, se ha incluído en el código PHP un manejo de errores muy simple en la gestión de las conexiones a la base de datos y de las operaciones sobre ella.

(*) Una herramienta genérica disponible como software libre y rápidamente aplicable, instalable en el mismo servidor del BotBasic como parte de cualquier BotBasic app, es Crudder. Ha sido creada por uno de los desarrolladores de BotBasic y está disponible en Internet. Hay, desde luego, otras opciones.

5. Crear en PHP las primitivas de BotBasic que acceden a las operaciones definidas en las clases

Una vez que la parte del modelo de negocios asociada a la información que se maneja está implementado en PHP, falta hacerlo con lo referente a la lógica conversacional. Esto requiere, primero, establecer vínculos de acceso por medio de los cuales el código en lenguaje BotBasic pueda interactuar con las clases de PHP recién diseñadas.

El quinto paso consiste en la creación en PHP de las "primitivas de BotBasic" asociadas a este modelo de negocio. Podemos observar que lo que hemos avanzado entre los pasos 1 y 4 no se relaciona aún con BotBasic en absoluto. Esto quiere decir que la lógica de negocios que hemos implementado en el paso 4 es la que permite implementar una web application en la intranet de la empresa para operaciones administrativas con manejo voluminoso de datos, por ejemplo.

Las primitivas son la capa de acceso al modelo de negocio que el lenguaje BotBasic requiere para conectar la lógica conversacional a los datos del modelo.

Más técnicamente, se puede decir que una primitiva no es más que un método de PHP dentro de una clase que implementa una "interfaz" llamada BizModelAdapterIface. Cada uno de estos métodos recibe una serie de argumentos como parámetros, efectúa operaciones sobre ellos, que frecuentemente son llamadas a las clases implementadas en el paso 4, y retorna valores de

respuesta en forma de arreglo de PHP.

Las primitivas que implementaremos acá no son más que un espejo de las operaciones del modelo de negocio. El código de PHP es fácil de entender:

```php
<?php

class BizModelAdapter implements BizModelAdapterIface
{

    // todos los metodos publicos deben retornar arreglos aunque devuelvan un unico valor

    static public function crearLote ()
    {
        return [ Lote::crear() ];
    }

    static public function solicitudesDeLote ($idLote)
    {
        return [ self::serializar(Lote::solicitudes($idLote)) ];
    }

    static public function recibirLote ($idLote, $idEmpleado)
    {
        Lote::recibir($idLote, $idEmpleado);
        return [];
    }

    static public function generarSolicitud ($idEmpleado, $idInsumo, $cantidad, $plazo, $observaciones)
    {
        return [ Solicitud::generar($idEmpleado, $idInsumo, $cantidad, $plazo, $observaciones) ];
    }

    static public function aprobarSolicitud ($idSolicitud, $idEmpleado)
    {
        Solicitud::aprobar($idSolicitud, $idEmpleado);
        return [];
    }

    static public function negarSolicitud ($idSolicitud, $idEmpleado)
    {
        Solicitud::negar($idSolicitud, $idEmpleado);
        return [];
    }

```

```php
    static public function despacharSolicitud ($idSolicitud)
    {
        Solicitud::despachar($idSolicitud);
        return [];
    }

    static public function incluirSolicitudEnLote ($idSolicitud, $idLote)
    {
        Solicitud::incluirEnLote($idSolicitud, $idLote);
        return [];
    }

    static public function solicitudesEnRango($diasMinimos, $diasMaximos)
    {
        return [ self::serializar(Solicitud::enRango($diasMinimos, $diasMaximos)) ];
    }

    static public function datosDeSolicitud ($idSolicitud)
    {
        return Solicitud::datos($idSolicitud);
    }

    static public function validarUsuario ($celular)
    {
        return [ Empleado::esValido($celular) ];
    }

    static public function puedeUsuarioAprobarNegar ($idEmpleado)
    {
        return [ Empleado::puedeAprobarNegar($idEmpleado) ? 1 : 0 ];
    }

    static public function puedeUsuarioRecibir ($idEmpleado)
    {
        return [ Empleado::puedeRecibir($idEmpleado) ? 1 : 0 ];
    }

    static public function listarCategorias ()
    {
        return Categoria::listar();
    }

```

```php
    static public function listarInsumos ($idCategoria)
    {
        return Insumo::listar($idCategoria);
    }

    static public function listarPlazos ()
    {
        return Solicitud::plazos();
    }

    static public function usuarioAprobador ($idEmpleado)
    {
        return [ Empleado::aprobador($idEmpleado) ];
    }

    static public function estadoSolicitud ($idSolicitud)
    {
        return [ Solicitud::estado($idSolicitud) ];
    }

    static public function loteEsValido ($idLote)
    {
        return [ Lote::esValido($idLote) ? 1 : 0 ];
    }

    static public function puedeUsuarioRecibirEsteLote ($idLote, $idEmpleado)
    {
        return [ Empleado::puedeRecibirLote($idEmpleado, $idLote) ? 1 : 0 ];
    }

    static private function serializar ($arreglo, $separador = '|')
    {
        return join($separador, $arreglo);
    }

}
```

Todo el código que se desarrolla en PHP, incluyendo el del paso anterior, debe ser validado antes de ponerlo en ejecución. Para ello es conveniente disponer de un Ambiente Integrado de Desarrollo (IDE) apropiado, como el gratuito *NetBeans*.

6. Desarrollar el programa de BotBasic

Hemos podido observar cómo en nuestra BotBasic app hay dos roles de usuario. Por un lado, los que trabajan en las unidades organizacionales solicitantes de insumos; allí hey varios subroles dependiendo de los "permisos" que los empleados tienen sobre las solicitudes (aprobar, negar, recibir). Por otro lado, la unidad de aprovisionamiento tiene un funcionamiento distinto: verifica las solicitudes y las encapsula en lotes antes de ordenar su despacho hacia las unidades solicitantes.

Los subroles han sido reflejados en el modelo de negocio, mientras que para el rol de los usuarios responsable en la unidad de aprovisionamiento definiremos una lógica conversacional distinta.

Así, habrá dos "bots" de BotBasic en este sistema:

1. BotSolic: bot usado por los usuarios de la unidad solicitante.
2. BotAprov: bot usado por los usuarios de la unidad de aprovisionamiento (en la que no hay subroles).

Ambos bots utilizan el mismo conjunto de primitivas desarrolladas anteriormente.

A continuación presentamos el código de BotBasic, producto de este sexto paso:

	A	B	C	D	E	F	G	H
1	**BOTBASIC 1.0 test code 1**							
2	MAGIC	BotBasic 1.0	El contenido de esta fila no debe ser modificado					
3	VERSION	1.0.0b	Versión inicial (beta)					
4	BOTS	bsolic baprov						
5								
6								
7								
8	**MESSAGES**							
9	name		es					
10								
11	celularAprovisionadorAutorizado		88996655					
12	enviaTuNumero		Escribe tu número de teléfono celular:					
13	usuarioInvalido		No está autorizado para utilizar esta app.					
14	optionSolicitarInsumo		Solicitar insumo					
15	optionAprobarSolicitud		Aprobar solicitud					
16	optionNegarSolicitud		Negar solicitud					
17	optionRecibirLote		Recibir lote					
18	introSolicitarInsumo		Efectuar solicitud de insumo dirigida a Departamento de Aprovisionamiento					
19	seleccioneCategoria		Seleccione la categoría del insumo para ver las posibilidades a elegir:					
20	seleccioneInsumo		Seleccione el insumo a solicitar:					
21	definaCantidad		Indique la cantidad del insumo:					
22	definaPlazo		Indique el plazo en que requiere que llegue el insumo:					
23	definaObservaciones		Agregue cualquier observación pertinente (una línea disponible):					
24	estaSeguro		¿Está seguro?					
25	si		Sí					
26	no		No					
27	solicitudRecienGenerada		Se ha generado la solicitud {idSolicitud}.					
28	solicitudEsperandoPorAprobacion		La solicitud está esperando por la aprobación del Jefe de la Unidad					
29	nuevaSolicitudPorAprobar		Nueva solicitud por aprobar: {idSolicitud}.					
30	introAprobarSolicitud		Aprobar una solicitud pendiente					
31	introNegarSolicitud		Negar una solicitud pendiente					
32	aprobar		APROBAR					
33	negar		NEGAR					
34	noPuedeAprobarOnegar		No está autorizado para aprobar o negar solicitudes.					
35	queAprobar		Indique cuál solicitud desea aprobar:					
36	optionUltimaSolicitud		La última solicitud reportada					
37	optionOtraSolicitud		Otra solicitud					
38	ingreseNumeroSolicitud		Ingrese el número de solicitud:					
39	solicitudNoPorAprobarOnegar		La solicitud indicada no está pendiente por ser aprobada o negada.					
40	solicitudAprobada		La solicitud ha sido aprobada.					
41	solicitudNegada		La solicitud ha sido negada.					
42	introRecibirLote		Recibir un lote de solicitudes del Departamento de Aprovisionamiento					
43	numeroDeLote		Ingrese el número de lote:					
44	loteInvalido		El número de lote indicado no es válido.					
45	contenidoDelLote		Solicitudes incluidas en el lote:					
46	pipeSpec		|					
47	noPuedeRecibirLote		No está autorizado para recibir este lote.					
48	resumenDatosSolicitud		{cantidad}x de {nombreCategoria} / {nombreInsumo}					
49	recibirElLote		¿Recibir el lote?					
50	loteRecibido		El lote ha sido recibido.					
51	opcionBuscarSolicitudes		Ubicar solicitudes					
52	opcionCrearLote		Crear lote de solicitudes					
53	opcionIncluirEnLote		Incluir solicitud en lote					
54	introBuscarSolicitudes		Ubicar solicitudes según fechas de entrega					
55	tipoDeBusqueda		Indique el tipo de búsqueda:					
56	rangoVencimiento		Indique el rango de búsqueda para las solicitudes vencidas:					
57	rangoDespacho		Indique el rango de búsqueda para las solicitudes a despachar:					
58	rangoDeBusquedaDiasInf		Límite inferior del rango (en días):					
59	rangoDeBusquedaDiasSup		Límite superior del rango (en días):					
60	resumenDatosSolicitudLargo		#{idSolicitud}: {cantidad}x de {NombreCategoria} / {nombreInsumo}					
61	introCrearLote		Crear nuevo lote de solicitudes para despacho unificado					
62	crearLote		CREARLOTE					
63	introIncluir		Incluir una solicitud en un lote existente					
64	incluir		INCLUIR					
65	numeroDeSolicitud		Ingrese el número de solicitud:					
66	solicitudInvalida		El número de solicitud indicado no es válido.					
67	fichaSolicitudCategoria		Categoria: {nombreCategoria}					
68	fichaSolicitudInsumo		Insumo: {nombreInsumo}					
69	fichaSolicitudCantidad		Cantidad: {cantidad}					
70	fichaSolicitudEstado		Estado: {estado}					
71	fichaSolicitudFecha		Fecha: {fecha}					
72	fichaSolicitudPlazo		Plazo de entrega solicitado: {plazo}					
73	fichaSolicitudObservaciones		Observaciones: {observaciones}					
74	fichaSolicitudUnidad		Unidad solicitante: {nombreUnidad}					
75	fichaSolicitudSolicitante		Persona que solicita: {nombreSolicitante}					
76	fichaSolicitudAprobador		Persona que aprueba la solicitud: {nombreAprobador}					
77	fichaSolicitudNegador		Persona que niega la solicitud: {nombreNegador}					
78	fichaSolicitudRecibidor		Persona que recibe: {nombreRecibidor}					
79	aprobado		aprobado					
80	imposibleUsarSolicitudParaLote		Debido a su estado, la solicitud no puede ser incluida en un lote					
81	loteCreado		El lote #{idLote} ha sido creado.					
82	solicitudIncluida		La solicitud ha sido incluida en el lote.					
83								
84								
85								
86	**MENUS**							
87	name		description		in	out	remarks	
88								
89								
90								
91	**MAGICVARS**							
92	name		description					
93								
94								
95								
96	**PRIMITIVES**							
97	name		description		in	out	remarks	
98								
99	CrearLote		Crea un nuevo lote y devuelte su id numérico recién creado por el RDBMS		--	intgIdLote		
100	SolicitudesDeLote		Recupera todos los id de las solicitudes relacionadas (incluidas) en el lote, o empty si el lote no tiene so		intgIdLote	strnIdsSolicitudes	Valor de retorno es idSolicitud1|idSolicitud2|...|idSolicitudN	

101	RecibirLote		Genera en la base de datos todas las relaciones Empleado-Solicitud de rol "recibe"	intgIdLote intgIdE	--		
102	GenerarSolicitud		Crea un nuevo registro de solicitud y devuelve su id recién creado por el RDBMS	intgIdEmpleado in	intgIdSolicitud	Cuarto argumento debe adoptar los valores del campo ENUM en la tabla de la BD	
103	AprobarSolicitud		Genera en la base de datos la relación Empleado-Solicitud de rol "aprueba"	intgIdSolicitud int	--		
104	NegarSolicitud		Genera en la base de datos la relación Empleado-Solicitud de rol "niega"	intgIdSolicitud int	--		
105	DespacharSolicitud		Asigna el estado "despachado por la unidad de aprovisionamiento" a la solicitud	intgIdSolicitud	--		
106	RecibirSolicitud		Genera en la base de datos la relación Empleado-Solicitud de rol "recibe"	intgIdSolicitud int	--		
107	IncluirSolicitudEnLote		Genera en la base de datos una relación Solicitud-Lote	intgIdSolicitud intg	--		
108	SolicitudesPendientes		Retorna un arreglo de id de solicitudes que deban ser entregados en la unidad solicitante dentro del rar	intgDiasMinimos i	strnIdsSolicitudes	\Valor de retorno es idSolicitud1|idSolicitud2|...|idSolicitudN	
109	DatosDeSolicitud		Devuelve la siguiente información asociada a la solicitud: nombre del insumo, nombre de la categoría d	intgIdSolicitud	strnNombreInsumo	strnNombreCategoria intgCantidad strnEstado strnFecha strnPlazo strnObservaciones strnNombreSo	
110	ValidarUsuario		Indica si un usuario puede o no usar el bot botsolic	strnCelular	boolEsValido	Los valores bool se retornan como enteros 0 o 1	
111	PuedeUsuarioAprobarNegar		Indica si un usuario puede aprobar o negar solicitudes	intgIdEmpleado	boolPuede		
112	PuedeUsuarioRecibir		Indica si un usuario puede recibir lotes provenientes de la Unidad de Aprovisionamiento	intgIdEmpleado	boolPuede		
113	ListarCategorias		Devuelve las categorias de insumos (nombre +hook de menu)	--	strnCat1 strnCat2	Retorna un número variable de valores, apto para a OPTIONS	
114	ListarInsumos		Devuelve los insumos asociados a una categoria (nombre +hook de menu)	intgIdCategoria	strnIns1 strnIns2	Retorna un número variable de valores, apto para a OPTIONS	
115	ListarPlazos		Devuelve los plazos de tiempo asociados a solicitudes	--	strnPla1 strnPla2	Retorna un número variable de valores, apto para a OPTIONS	
116	UsuarioAprobador		Retorna el usuario al que le toca aprobar las solicitudes del usuario indicado como argumento	intgIdEmpleado	intgIdAprobador		
117	EstadoSolicitud		Devuelve el estado de una solicitud	intgIdSolicitud	strnEstado		
118	LoteEsValido		Indica si un número de lote es o no válido	intgIdLote	--		
119	PuedeUsuarioRecibirEsteLote		Indica si un usuario puede recibir un lote específico	intgIdLote intgIdE	boolPuede		
120							
121							
122							
123	**PROGRAM**						
124	botsolic	botsolic	botsolic	botaprov	botaprov	botaprov	
125							
126	ENTRYHOOK		REM validar que el usuario este conectado; si no lo esta, pedir su numero de celular	ENTRYHOOK		REM validar que el usuario este conectado; si no lo esta, pedir su numero de celular	
127			USERID TO sId			USERID TO sId	
128			IF NOT EMPTY sId THEN END			IF NOT EMPTY sId THEN END	
129			INPUT cellphone TITLE enviaTuNumero TO sCelular			INPUT cellphone TITLE enviaTuNumero TO aCelular	
130			CALL ValidarUsuario sCelular TO sId			IF NEQ aCelular celularAprovisionadorAutorizado THEN PRINT usuarioInvalido : RESET ALL : END	
131			IF EQ sId -1 THEN PRINT usuarioInvalido : RESET ALL : END			USERID FROM aCelular	
132			USERID FROM sId			END	
133			END				
134				/start /inicio	start	REM menu principal	
135	/start /inicio	start	REM menu principal			RESET ALL	
136			RESET ALL			USERID TO sId	
137			USERID TO sId			MENU OPTIONS opcionBuscarSolicitudes opcionCrearLote opcionIncluirEnLote TO opcion	
138			CALL PuedeUsuarioAprobarNegar sId TO puedeAprobarNegar			IF EQ opcion opcionBuscarSolicitudes THEN GOTO buscar	
139			CALL PuedeUsuarioRecibir sId TO puedeRecibir			IF EQ opcion opcionCrearLote THEN GOTO crearLote	
140			OPTIONS optionSolicitarInsumo			IF EQ opcion opcionIncluirEnLote THEN GOTO incluir	
141			IF EQ puedeAprobarNegar 1 THEN OPTIONS optionAprobarSolicitud optionNegarSolicitud			END	
142			IF EQ puedeRecibir 1 THEN OPTIONS optionRecibirLote				
143			MENU TO opcion	/buscar	buscar	REM buscar solicitudes	
144			IF EQ opcion optionSolicitarInsumo THEN GOTO solicitar			PRINT introBuscarSolicitudes	
145			IF EQ opcion optionAprobarSolicitud THEN GOTO aprobar			OPTIONS optionSolicitudesVencidas optionSolicitudesPendientes	
146			IF EQ opcion optionNegarSolicitud THEN GOTO negar			MENU TITLE tipoDeBusqueda TO opcion	
147			IF EQ opcion optionRecibirLote THEN GOTO recibir			IF EQ opcion optionSolicitudesVencidas THEN SET k -1 ELSE SET k 1	
148			END			IF EQ opcion optionSolicitudesVencidas THEN PRINT rango\Vencimiento ELSE PRINT rangoDespacho	
149						INPUT date TITLE rangoDeBusquedaDiasInf TO dias1	
150	/solicitar	solicitar	REM solicitar un insumo			INPUT date TITLE rangoDeBusquedaDiasSup TO dias2	
151			PRINT introSolicitarInsumo			IF GT dias1 dias2 THEN SET tmp dias1 : SET dias1 dias2 : SET dias2 tmp	
152			CALL ListarCategorias TO OPTIONS			IF EQ k -1 THEN SET tmp dias1 : SET dias1 dias2 : SET dias2 tmp : MUL dias1 -1 : MUL dias2 -1	
153			MENU TITLE seleccioneCategoria PAGER pagerShort 10 TO idCategoria			CALL SolicitudesPendientes dias1 dias2 TO idsSolicitudes	
154			CALL ListarInsumos idCategoria TO OPTIONS			REM loop sobre los ids, que vienen como id1|id2|id3|...|idN	
155			MENU TITLE seleccioneInsumo PAGER pagerShort 10 TO idInsumo			SET aSeparar idsSolicitudes	
156			INPUT integer TITLE definaCantidad TO cantidad			PRINT contenidoDelLote	
157			CALL ListarPlazos TO OPTIONS		buscar1a	SPLIT pipeSpec aSeparar TO idSolicitud aSeparar	
158			MENU TITLE definaPlazo TO plazo			IF EMPTY idSolicitud THEN GOTO buscar1b	
159			INPUT string TITLE definaObservaciones TO observaciones			CALL DatosDeSolicitud idSolicitud TO insumoNombre categoriaNombre cantidad estado fecha plazo observaciones nombreSolicitante nombreAprobador nombreNegador nombreRecibidor nombreUnidad	
160			MENU TITLE estaSeguro OPTIONS si no TO confirmar			PRINT resumenDatosSolicitudLargo	
161			IF EQ confirmar no THEN GOTO start			GOTO buscar1a	
162			CALL GenerarSolicitud sId idInsumo cantidad plazo observaciones TO idSolicitud		buscar1b	GOTO start	
163			PRINT solicitudRecienGenerada solicitudEsperandoPorAprobacion				
164			CALL UsuarioAprobador sId TO idAprobador	/crearlote	crearLote	REM crear un lote con una solicitud inicialmente	
165			PRINT nuevaSolicitudPorAprobar ON botsolic idAprobador			PRINT introCrearLote	
166			SET ultimaSolicitudCreada idSolicitud ON botsolic idAprobador			GOSUB crearoincluir crearLote -1	
167			GOTO start			GOTO start	
168							
169	/aprobar	aprobar	REM aprobar una solicitud	/incluir	incluir	REM incluir una solicitud en un lote	
170			PRINT introAprobarSolicitud			PRINT introIncluir	
171			GOSUB aprobaronegar aprobar			INPUT integer TITLE numeroDeLote TO idLote	
172			GOTO start			CALL LoteEsValido idLote TO esValido	
173						IF EQ esValido 0 THEN PRINT loteInvalido : GOTO start	
174	/negar	negar	REM negar una solicitud			GOSUB crearoincluir incluir idLote	
175			PRINT introNegarSolicitud			GOTO start	
176			GOSUB aprobaronegar negar				
177			GOTO start		crearoincluir	REM crear un lote o incluir una solicitud en un lote (rutina auxiliar)	
178						ARGS accion argIdLote	
179		aprobarnegar	REM aprobar o negar una una solicitud (rutina auxiliar)			INPUT integer TITLE numeroDeSolicitud TO idSolicitud	
180			ARGS accion			CALL DatosDeSolicitud idSolicitud TO nombreInsumo nombreCategoria cantidad estado fecha plazo observaciones nombreSolicitante nombreAprobador nombreNegador nombreRecibidor nombreUnidad	
181			CALL PuedeUsuarioAprobarNegar sId TO puedeAprobarNegar			IF EMPTY nombreInsumo THEN PRINT solicitudInvalida : RETURN	
182			IF EQ puedeAprobarNegar 0 THEN PRINT noPuedeAprobarOnegar : GOTO start			PRINT fichaSolicitudCategoria fichaSolicitudInsumo fichaSolicitudCantidad fichaSolicitudEstado	
183			IF EMPTY ultimaSolicitudCreada THEN GOTO aprobarnegar1			PRINT fichaSolicitudFecha fichaSolicitudPlazo fichaSolicitudObservaciones fichaSolicitudUnidad	
184			MENU TITLE queAprobar OPTIONS optionUltimaSolicitud optionOtraSolicitud TO opcion			PRINT fichaSolicitudSolicitante fichaSolicitudAprobador fichaSolicitudNegador fichaSolicitudRecibidor	
185			IF EQ opcion optionUltimaSolicitud THEN SET idSolicitud ultimaSolicitudCreada : GOTO aprobarnegar2			IF NEQ estado aprobado THEN PRINT imposibleUsarSolicitudParaLote : RETURN	
186		aprobarnegar1	INPUT integer TITLE ingreseNumeroSolicitud TO idSolicitud			IF EQ accion crearLote THEN CALL CrearLote TO idLote ELSE SET idLote argIdLote	
187			CALL EstadoSolicitud idSolicitud TO estado			CALL IncluirSolicitudEnLote idSolicitud idLote	
188			IF NEQ estado solicitado THEN PRINT solicitudNoPorAprobarOnegar : GOTO aprobarnegar1			IF EQ accion crearLote THEN PRINT loteCreado ELSE PRINT solicitudIncluida	
189		aprobarnegar2	REM ya se dispone del numero de solicitud a aprobar o negar en idSolicitud			RETURN	
190			MENU TITLE estaSeguro OPTIONS si no TO confirmar				
191			IF EQ confirmar no THEN GOTO start				
192			IF EQ accion aprobar THEN CALL AprobarSolicitud idSolicitud sId : PRINT solicitudAprobada				
193			IF EQ accion negar THEN CALL NegarSolicitud idSolicitud sId : PRINT solicitudNegada				
194			RETURN				
195							
196	/recibir	recibir	REM recibir un lote de solicitudes				
197			PRINT introRecibirLote				
198			INPUT integer TITLE numeroDeLote TO idLote				

199		CALL LoteEsValido idLote TO esValido	
200		IF EQ esValido 0 THEN PRINT loteInvalido : GOTO start	
201		CALL SolicitudesDeLote TO idsSolicitudes	
202		REM loop sobre los ids, que vienen como id1|id2|id3|...|idN	
203		SET aSeparar idsSolicitudes	
204		PRINT contenidoDelLote	
205	recibir1a	SPLIT pipeSpec aSeparar TO idSolicitud aSeparar	
206		IF EMPTY idSolicitud THEN GOTO recibir1b	
207		CALL DatosDeSolicitud idSolicitud TO nombreInsumo nombreCategoria cantidad estado fecha plazo observaciones nombreSolicitante nombreAprobador nombreNegador nombreRecibidor nombreUnidad	
208		CALL PuedeUsuarioRecibirEsteLote isLote sId TO puedeRecibir	
209		IF EQ puedeRecibir THEN PRINT noPuedeRecibirLote : GOTO start	
210		PRINT resumenDatosSolicitud	
211		GOTO recibir1a	
212	recibir1b	MENU TITLE recibirElLote OPTION si no TO recibir	
213		IF EQ recibir no THEN GOTO start	
214		CALL recibirLote idLote sId	
215		PRINT loteRecibido	
216		GOTO start	
217			

7. Instalar y compilar el código de BotBasic

Hemos desarrollado el código en BotBasic, pero no sabemos aún si funciona.

Verificar la correctitud de la solución se hace por medio de dos actividades consecutivas. La primera es la compilación del código de BotBasic desarrollado. Esto se puede hacer por medio de una interfaz web o desde la línea de comandos del servidor de BotBasic, así:

```
$ botbasicparser --filename insumosbbcode.csv --codename insumosbbapp
```

Aquí, *insumosbbcode.csv* es el contenido del archivo de código de BotBasic editado en la aplicación de hoja de cálculo y exportado en formato CSV, mientras que *insumosbbapp* es el nombre de la BotBasic app.

La resultante de este comando es un indicador de éxito o falla en la compilación. Cuando hay falla, el compilador arroja un detalle de los errores encontrados para cada sección y número de línea (número de fila, en la hoja de cálculo) del programa de BotBasic.

Cuando el programa compila correctamente, se debe configurar luego un archivo con la especificación de los bots de la chatapp (previamente creados por medio de la respectiva app) que serán utilizados por la BotBasic app. En el caso de Telegram, cada bot o conjunto de bots de Telegram serán asociados a bots del programa de BotBasic (BS, BA).

Una vez efectuada esta configuración, es la hora de probar la solución "en caliente", como se hace para toda solución de software. Durante de la fase de prueba serán detectados errores en la lógica de las primitivas y del código de BotBasic. Muchos de estos errores podrán ser detectados por medio de la bitácora de eventos de BotBasic, la cual se refleja en archivos de *log* o en una tabla en la base de datos SQL.

La puesta en producción de la BotBasic app desarrollada se efectúa cuando se ha validado la correctitud de la implementación.

FIN DEL DOCUMENTO