**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.

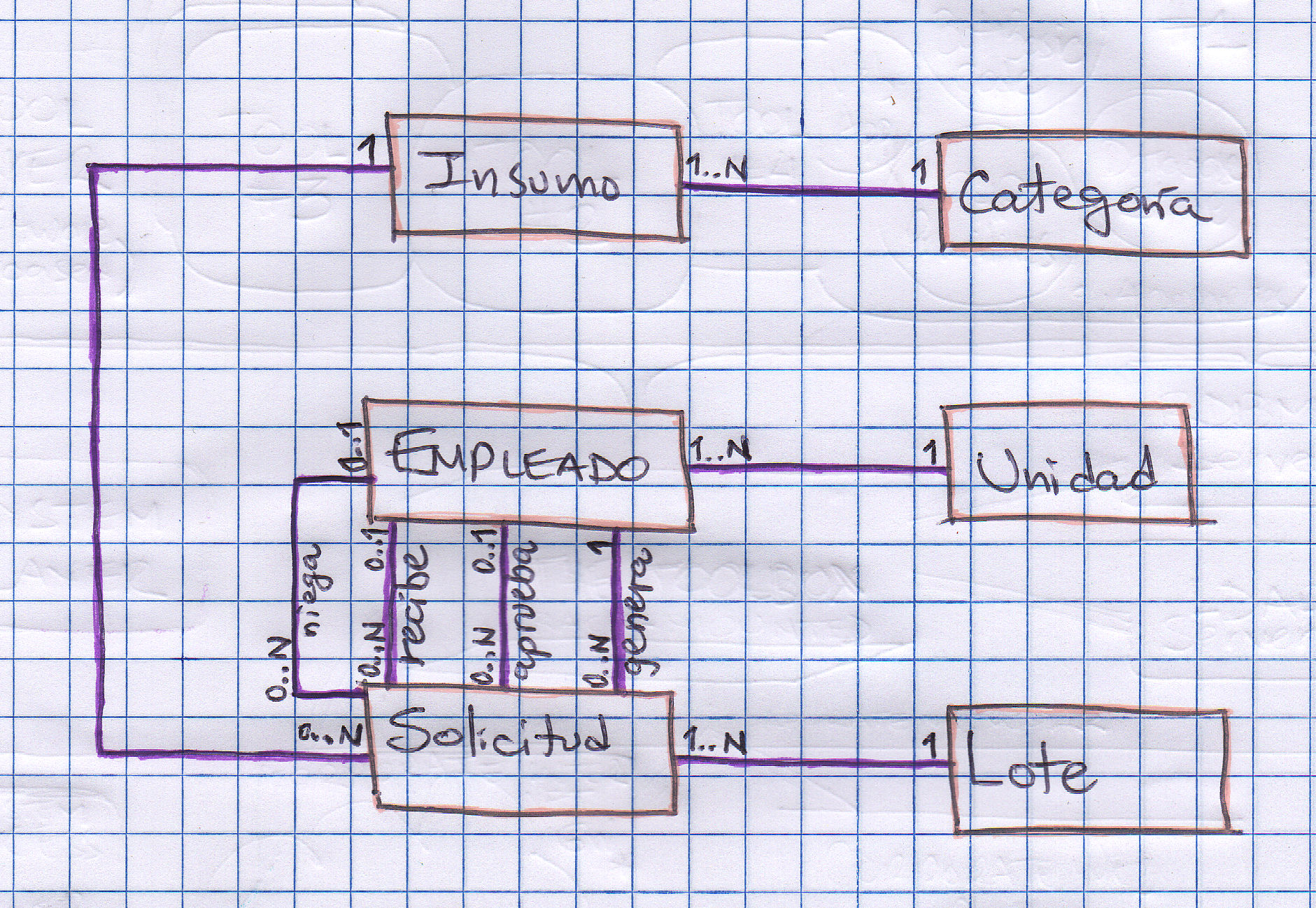
2. Crear el modelo de datos

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: | Categoría: |
| Unidad: |
| Empleado: |

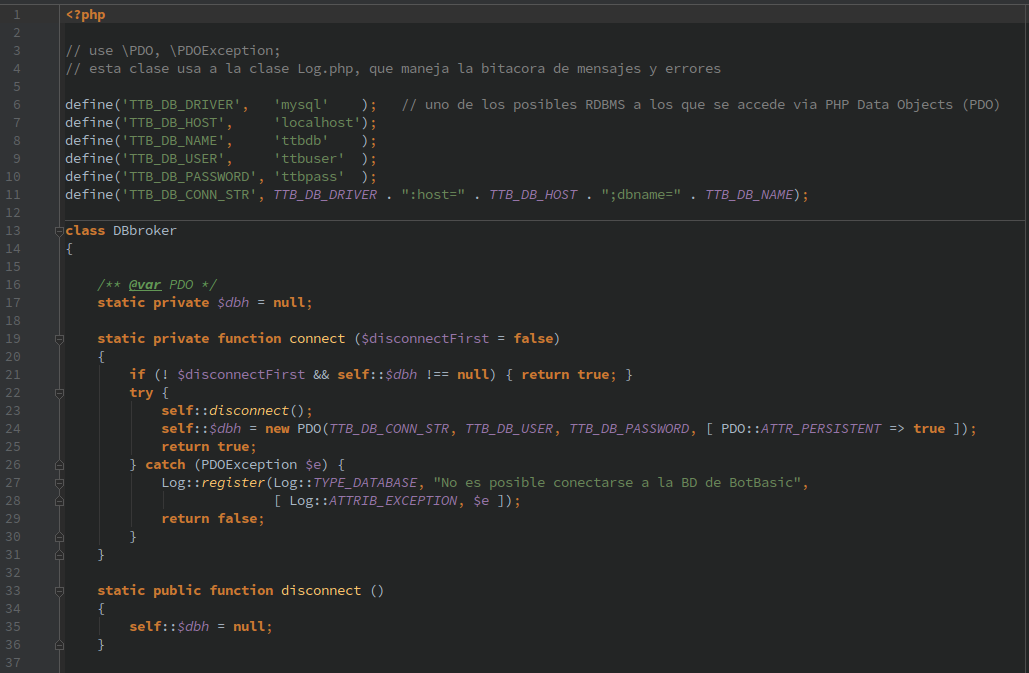
4. Crear las clases en PHP que acceden a los datos almacenados en las tablas

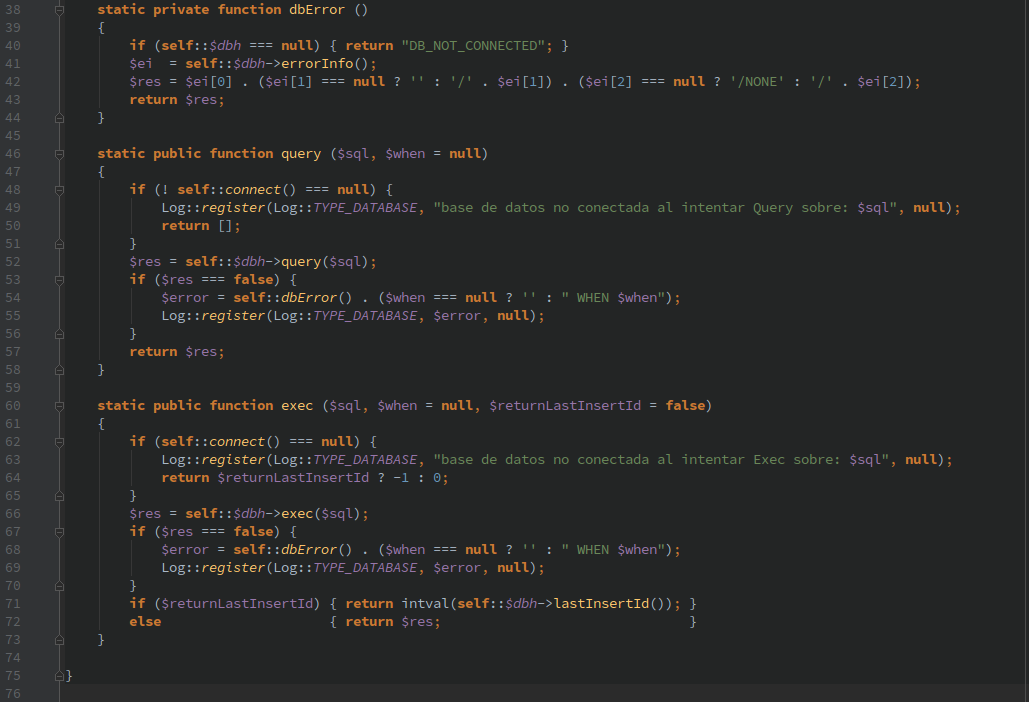
Normalmente, sobre las filas (registros) de las tablas se aplican operaciones tipo CRUD (create, read, update y delete). No siempre es necesario implementar todas las operaciones. Nuestra BotBasic app, por ejemplo, no requiere borrado de registros, pero sí creación, actualización y lectura.

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:



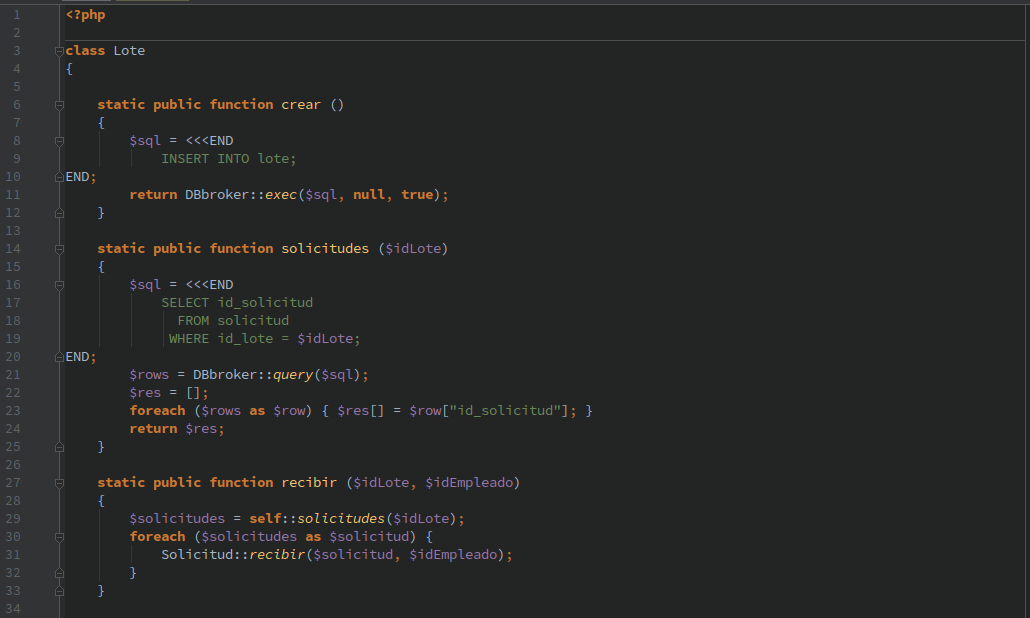


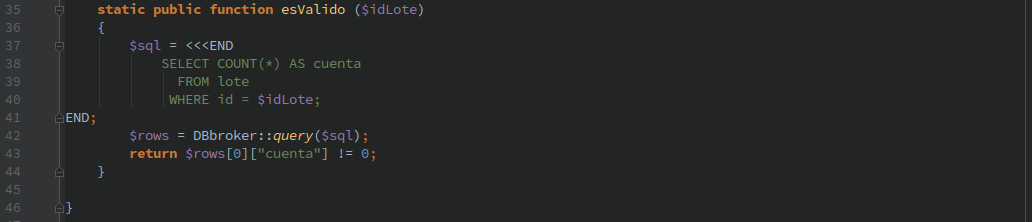
Empezaremos ahora con las operaciones más importantes asociadas a la entidad Lote:

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:

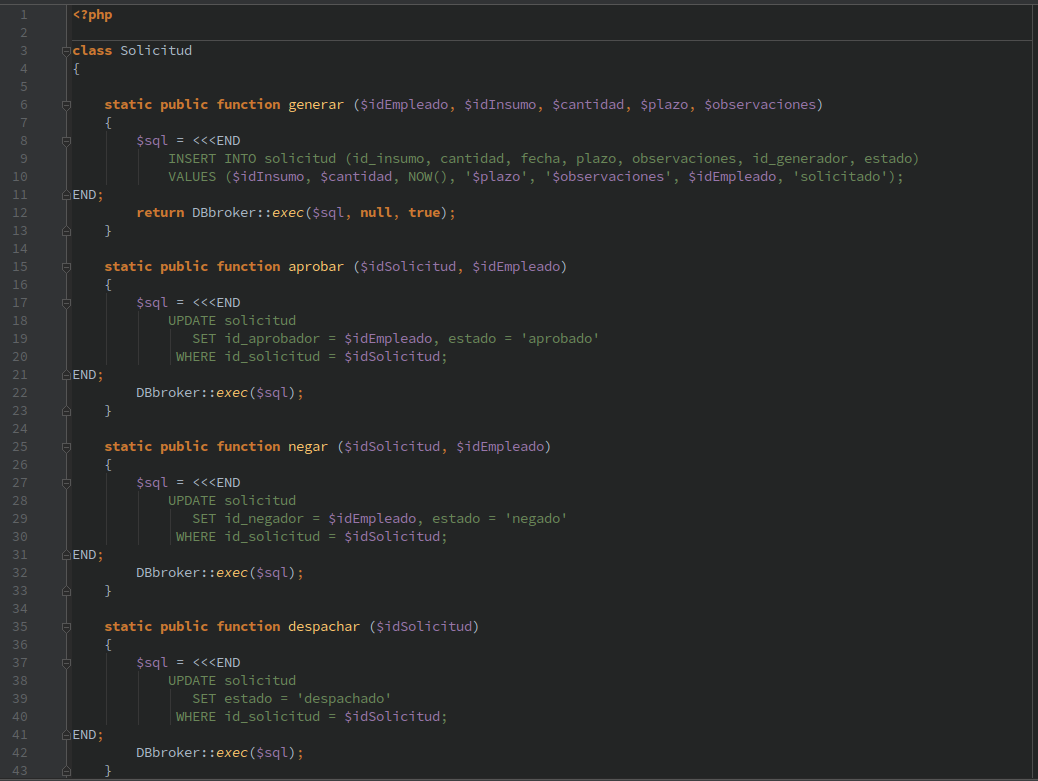


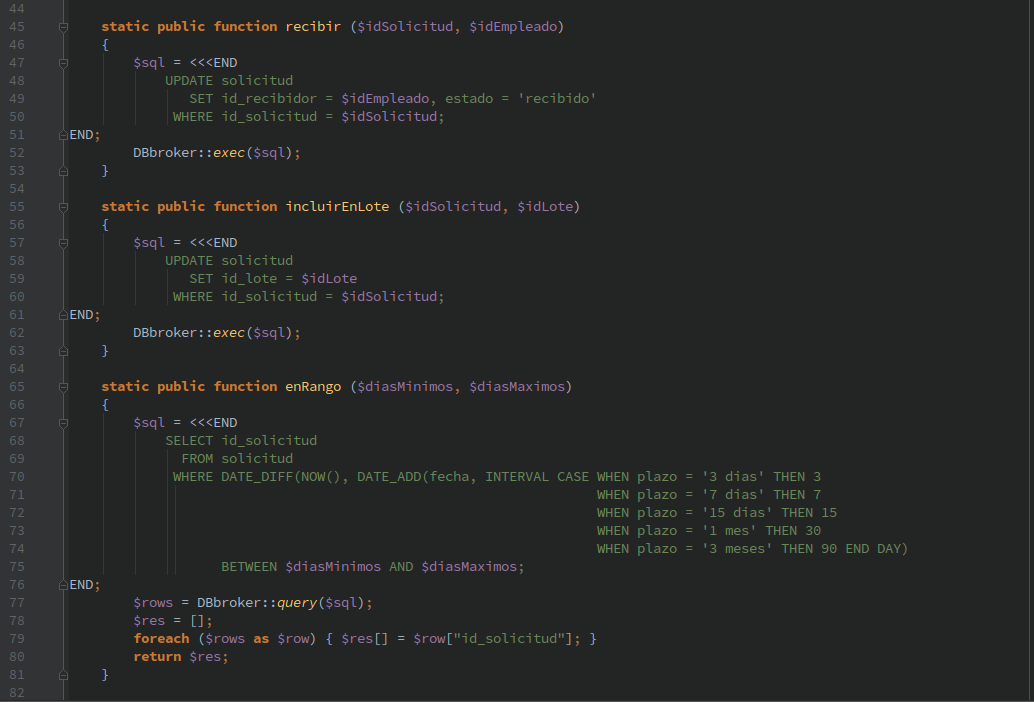


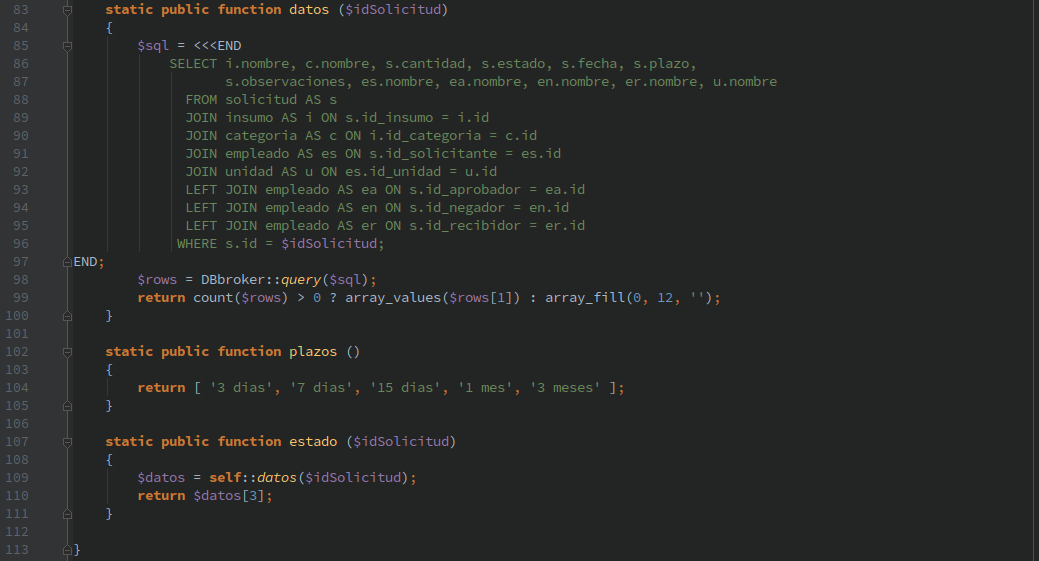
Estas son las operaciones más importantes asociadas a la entidad Solicitud:

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:

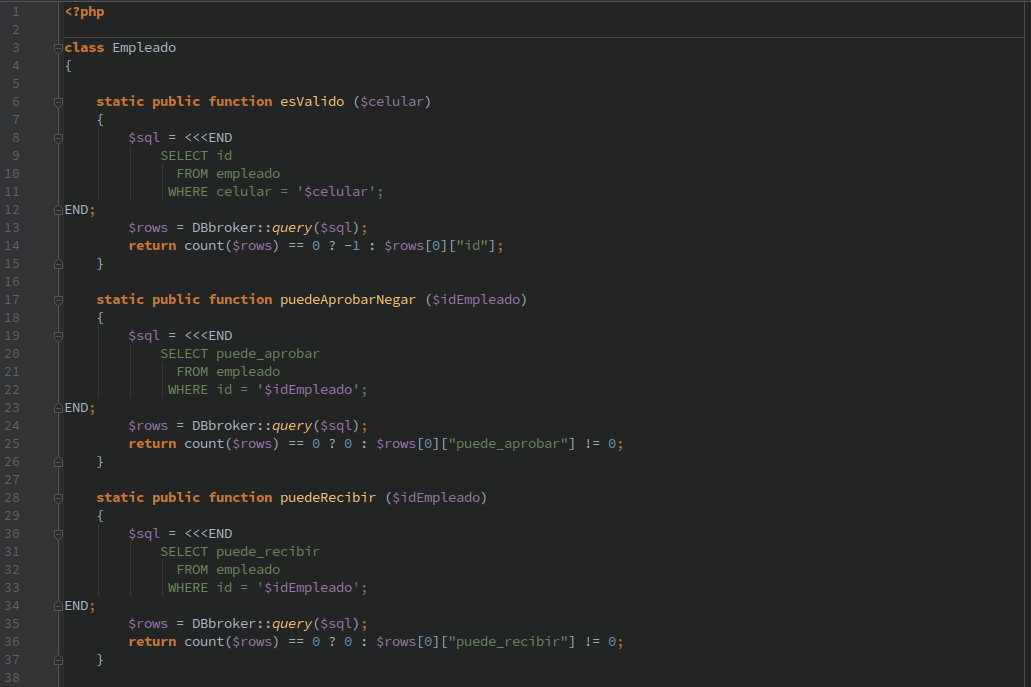


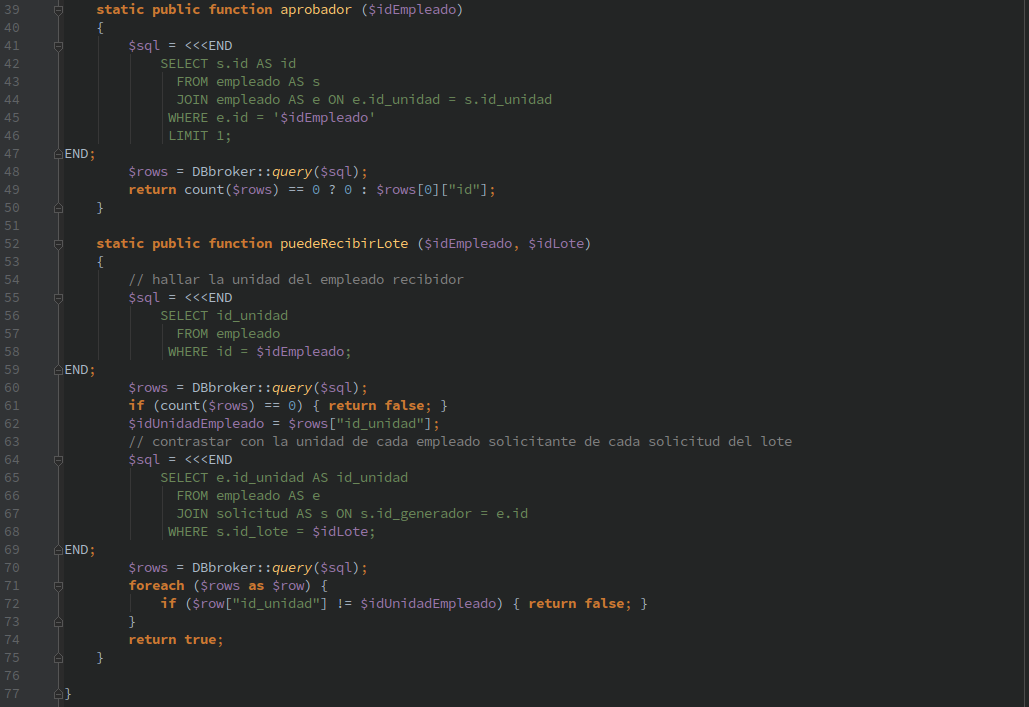




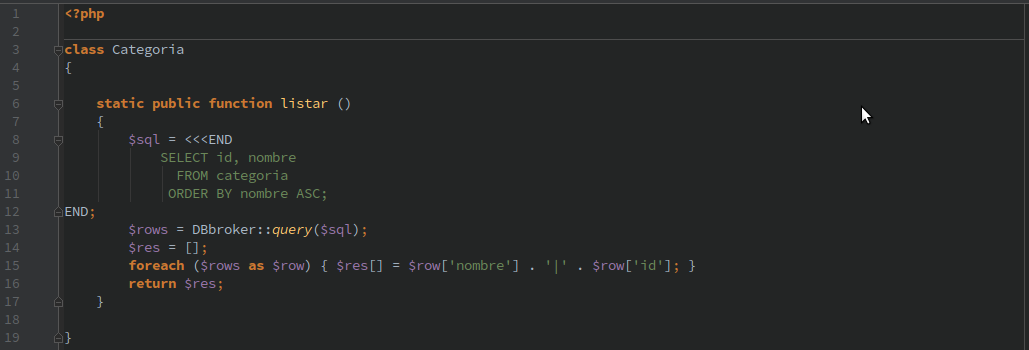
Definiremos también operaciones auxiliares sobre las entidades Empleado, Categoría e Insumo:

Para Empleado:

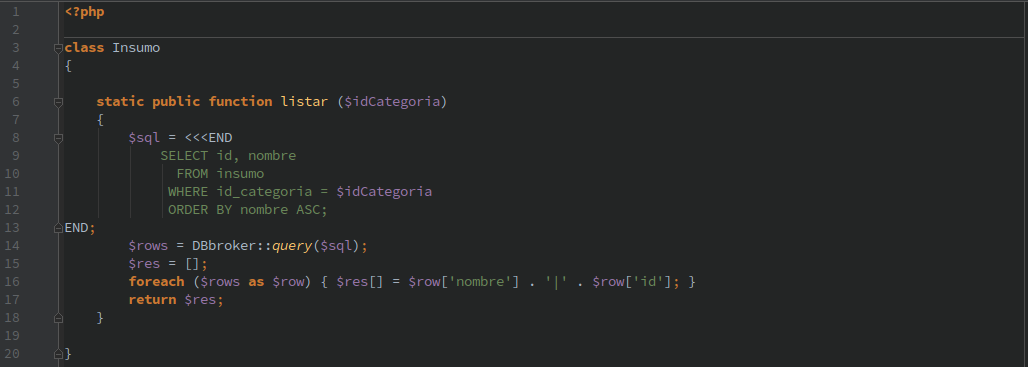




Para Categoria:



Para Insumo:



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

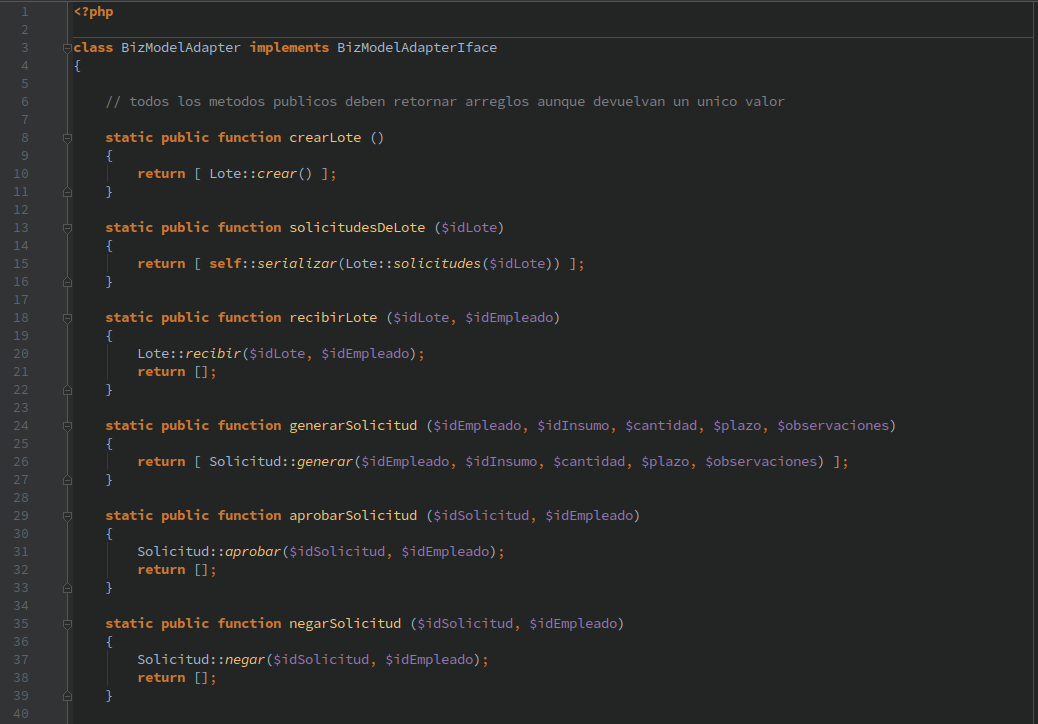
Una vez que la parte del modelo de negocios asociada a la información que se maneja está implementado en PHP, falta hacerlo con lo referente a la lógica conversacional. Esto requiere, primero, establecer vínculos de acceso por medio de los cuales el código en lenguaje BotBasic pueda interactuar con las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

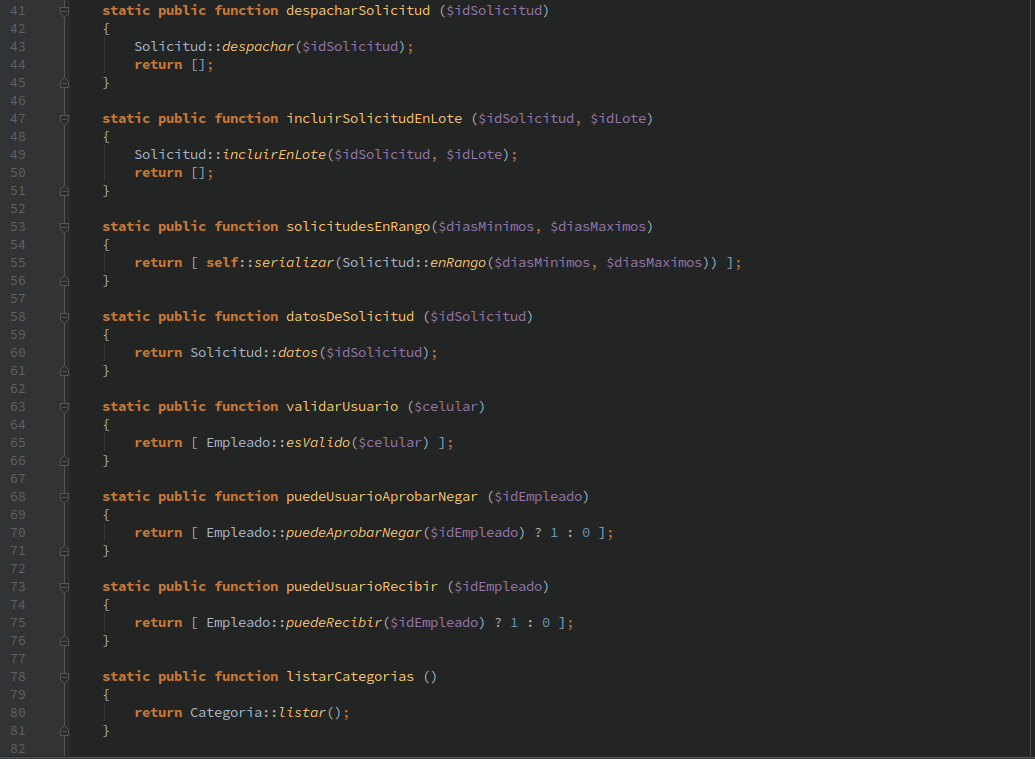
El quinto paso consiste en la creación en PHP de las "primitivas de BotBasic" asociadas a este modelo de negocio. Podemos observar que lo que hemos avanzado entre los pasos 1 y 4 no se relaciona aún con BotBasic en absoluto. Esto quiere decir que la lógica de negocios que hemos implementado en el paso 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:







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.

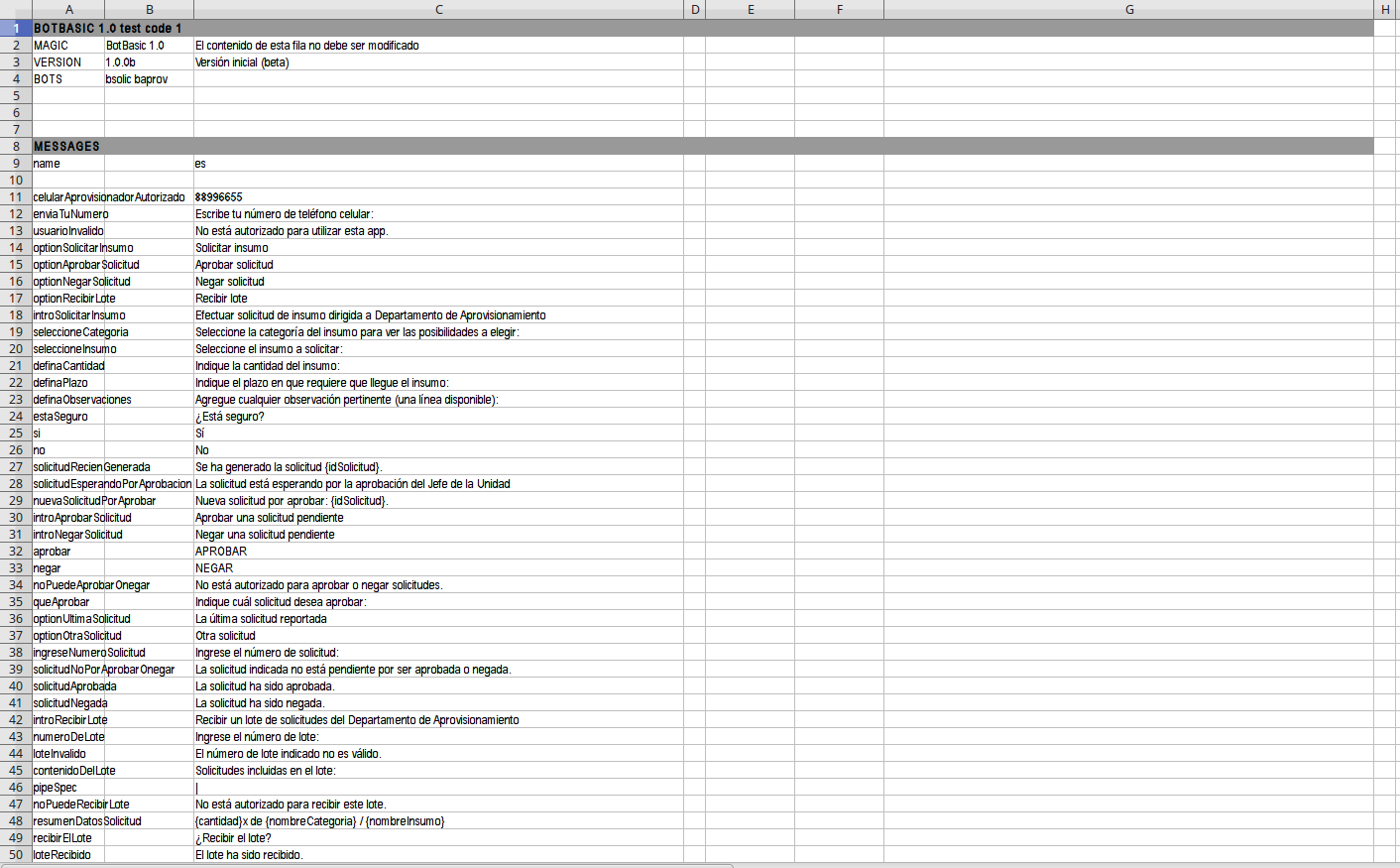
Los subroles han sido reflejados en el modelo de negocio, mientras que para el rol de los usuarios responsable en la unidad de aprovisionamiento definiremos una lógica conversacional distinta.

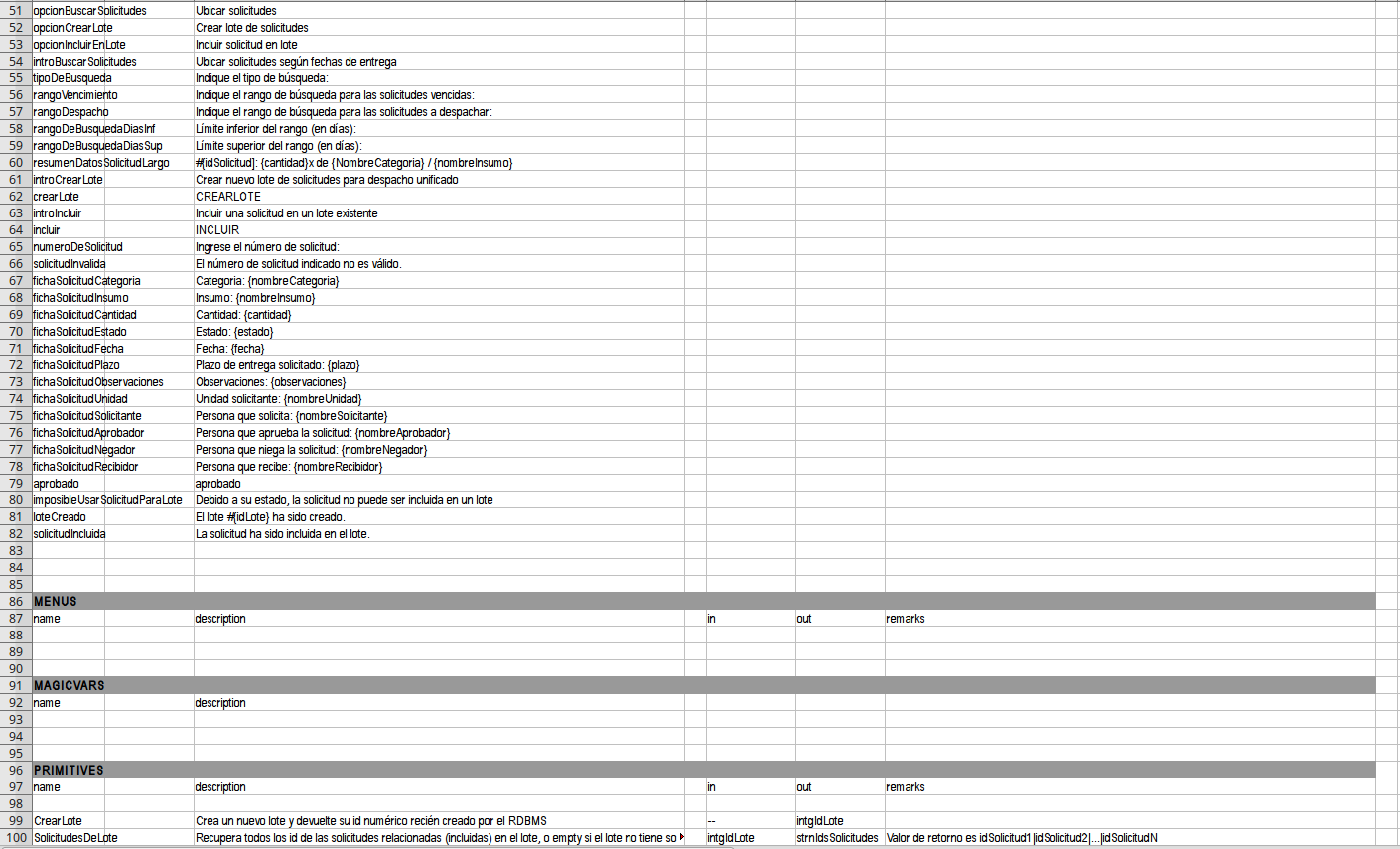
Así, habrá dos "bots" de BotBasic en este sistema:

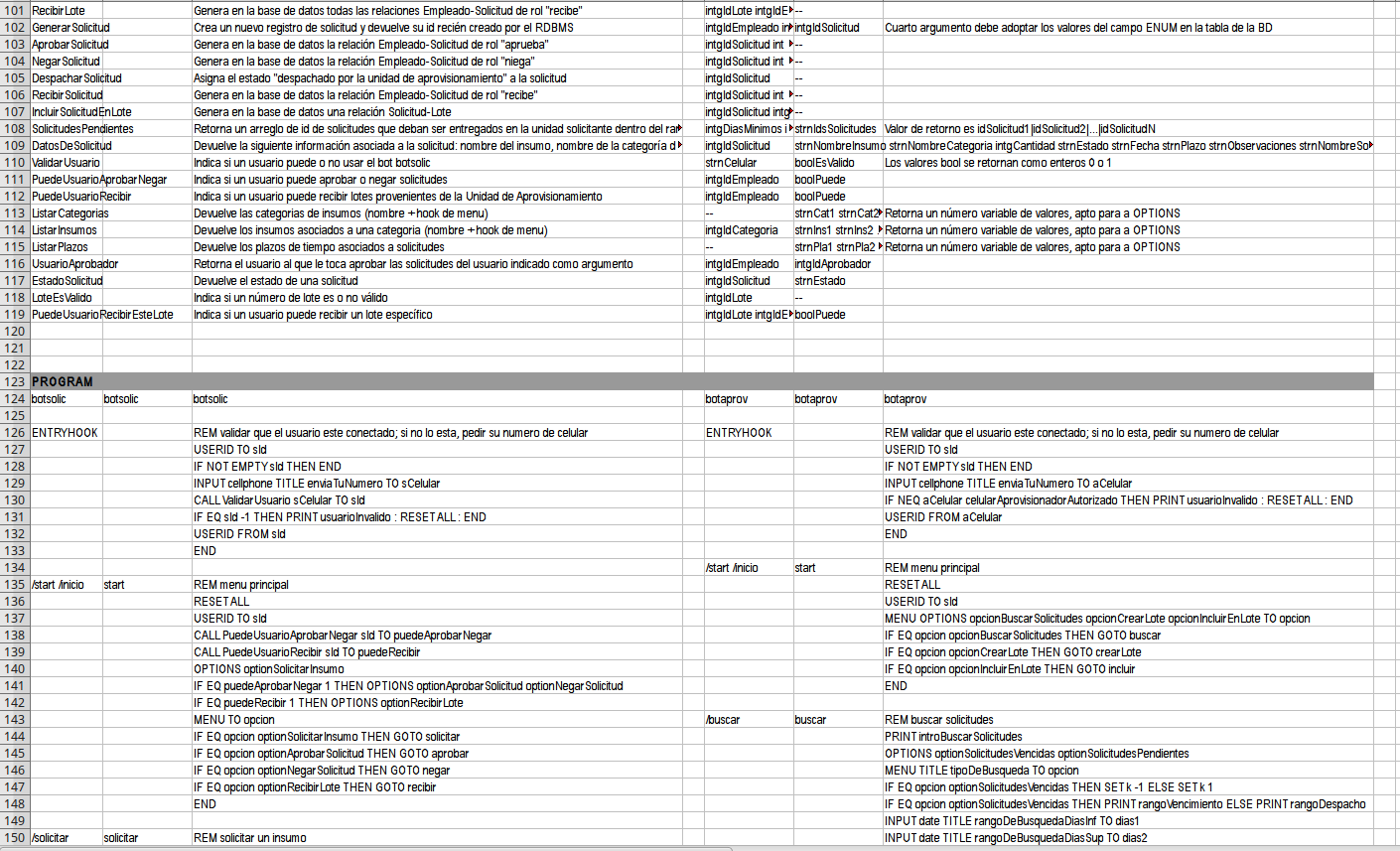
1. BotSolic: bot usado por los usuarios de la unidad solicitante.
2. BotAprov: bot usado por los usuarios de la unidad de aprovisionamiento (en la que no hay subroles).

Ambos bots utilizan el mismo conjunto de primitivas desarrolladas anteriormente.

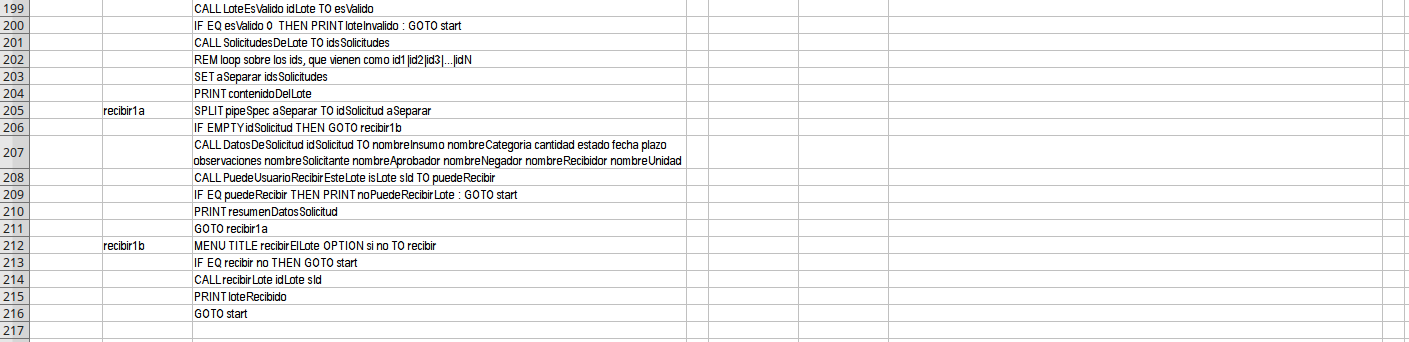
A continuación presentamos el código de BotBasic, producto de este sexto paso:











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