Implementación de nuevas prestaciones dinámicas al Centro Virtual de Información Tecnológica del Azúcar

Joel Arcia Montes de Oca.

Instituto Cubano de Investigaciones de los Derivados de la Caña de Azúcar (ICIDCA). Vía Blanca # 804, esq. a Carretera Central. San Miguel del Padrón. La Habana 11000. Cuba.

Joel.arcia@icidca.azcuba.cu

RESUMEN:

En el contexto actual, la Gestión de la Información y el Conocimiento, requiere de plataformas como los *CMS*[1]. En múltiples ocasiones estos sistemas no brindan respuesta a todos los requerimientos de los clientes y se hace necesario la construcción de *CMS* personalizados, que dispongan de funcionalidades específicas. Este trabajo tiene como objetivo desarrollar en el software Cevita (Centro Virtual de Información Tecnológica del Azúcar), aplicación que actualmente se encuentra en su etapa de validación, dos nuevas prestaciones que potencien su uso, y que lo aproximen a las características de un *CMS*. Para este fin se utilizó el *Framework* Symfony 1.4, herramienta de Código Abierto que facilita, la generación de código fuente, la creación de módulos básicos de administración y la extensión y personalización de estos. Finalmente se logró la implementación de dos nuevos módulos autónomos, el primero, sin necesidad de un programador, permite a los editores definir diferentes categorías, y dentro de cada una de éstas, los recursos externos (enlaces y alimentadores *RSS*[2] de sitios web de

[1] Sistemas de Gestión de Contenidos (Content Management Systems, CMS por sus siglas en inglés).
[2] Sindicación Simple de Contenidos (Really Simple Syndication, RSS por sus siglas en inglés).

interés) que corresponden, y el segundo, es un Sistema Autónomo de Publicación de Boletines. Las nuevas características desarrolladas, logran incrementar considerablemente la velocidad de publicación y gestión, así como, brindan un acceso confiable y oportuno a la información científica, disponiendo de esta, en el momento y lugar convenientes. Los resultados alcanzados pueden servir de referencia a desarrolladores que pretendan incluir en sus plataformas características dinámicas de la Web 2.0.

Palabras Claves: Gestión de la Información y el Conocimiento, Sistemas de Gestión de Contenidos (*CMS*), Código Abierto, *Framework* Symfony, *alimentadores RSS*, Web 2.0.

ABSTRACT:

Nowadays, information and knowledge management requires different applications/systems like CMS[3]. In multiple occasions, these systems do not provide an optimal solution of all client's requirements and therefore it is necessary the construction of customized CMS that includes specific functionalities. Particularly, this paper aims to developing two new-brand features within the Cevita's software (Virtual Center of Sugar Technology Information) which is currently at validation stage. Such features will significantly enhance the user/client software interaction, as well as it will bring it closer to the CMS characteristics. For this purpose, an Open Source tool named "Symfony Framework 1.4" was used. This tool facilitates the source code generation, the creation of basic administration modules which in turn can be extended and customized. These two autonomous modules were successfully implemented. The first one, where the presence of a computer programmer is not required, allows editors to define several categories and the corresponding external resources (links and Feed RSS[4] of relevant websites) within each of them. Likewise, the second one is an Autonomous Bulletin Publication System. The

[3] Content Management Systems.
[4] Really Simple Syndication.

new developed features considerably increase the publication and management speed, as well as provide reliable and timely access to scientific information. The achieved results might be used as an initial reference by developers who intend to include Web 2.0 dynamic features in their systems.

Keywords: Information and knowledge management, CMS, Open Source, Symfony Framework, Feed RSS, Web 2.0.

INTRODUCCIÓN.

El “contenido” se encuentra en un nivel de confiabilidad superior a la “información” y al “dato”. Este es creado a través de un proceso editorial, conformado por las acciones que se realizan en aras de preparar la información para su publicación a una audiencia, entre las que se encuentran: revisar, editar, comparar, aprobar, entre otras. El “contenido” es básicamente, la información utilizada con un propósito determinado y está constantemente sujeto a evaluación e interpretación. El proceso editorial referido es imperfecto y variable, no existen tecnologías o aplicaciones informáticas que garanticen la calidad del contenido que almacenan y gestionan. En este contexto, el ser humano sigue siendo el principal actor y responsable. Sin embargo, los sistemas informáticos juegan un papel esencial en la gestión óptima de contenidos y en la generación de conocimiento. (1)

La gestión de contenidos ha sido una necesidad presente desde que los seres humanos comenzaron a generarlos, incluso antes de la era digital. La búsqueda de soluciones en este sentido ha sido una constante a través de los tiempos. Hoy en día, el incremento considerable del flujo de información, la necesidad de estudiarla, editarla y publicarla como contenido útil, se ha acrecentado considerablemente, lo que ha conllevado al desarrollo e implementación de Sistemas Automatizados de Gestión de Información y Conocimiento más eficaces. Con el

propósito de implementar con mayor calidad, confiabilidad y celeridad este tipo de aplicaciones en los diferentes ámbitos, especialmente en el científico y académico, se han desarrollado los llamados *CMS*.

Los *CMS* son aplicaciones informáticas, que como su nombre indica, permiten gestionar contenidos, permitiendo a los editores cambiar texto, imágenes, videos o cualquier tipo de documento, así como clasificarlos y publicarlos. Todo sin necesidad de enfrentarnos al código fuente del software. Generalmente los contenidos gestionados en estos sistemas se almacenan en bases de datos que se actualizan empleando una interfaz web, accedida a través del navegador. En su mayoría, los *CMS* están compuestos por dos escenarios, el *Backend*, donde acceden los administradores y editores con roles y permisos especiales para la gestión de la información, incluyendo la gestión de usuarios del sistema, y el *Frontend*, donde acceden los usuarios comunes para visualizar los contenidos publicados. Estos también se definen, como un conjunto de procedimientos utilizados para describir procesos en un ambiente que requiere colaboración entre los diferentes actores. (2)(3)

Las ventajas que representan el uso de los *CMS* en la creación de aplicaciones basadas en tecnología web y bases de datos son innumerables. Van desde el ahorro de tiempo y trabajo, al incorporar por defecto tareas repetitivas de administración[5], hasta la construcción de un sistema de seguridad basado en perfiles de usuarios. También facilitan la selección e implementación de un diseño apropiado, adaptable a los múltiples dispositivos existentes. Por otra parte, viabilizan los procesos de clasificación y publicación periódica. Todo lo anterior, realizado mediante interfaces basada en formularios web, accedidas desde un navegador, sin necesidad de dominar la programación. (4)

[5] Crear, Consultar, Actualizar y Eliminar (CRUD, por sus siglas en inglés)

Igualmente, el uso de *CMS* tiene sus desventajas. Por ejemplo, al instalar un sistema de este tipo, se incluyen muchos componentes y módulos por defecto, que no son de interés para el cliente, por lo que se hará un poco más lenta la navegación, sobre todo cuando no se cuenta con una infraestructura tecnológica de gama alta. En múltiples ocasiones estas plataformas se encuentran disponibles de forma gratuita en Internet, pero generalmente, las nuevas prestaciones no son gratuitas, adaptables y escalables, o su adaptabilidad y escalabilidad presentan un alto grado de complejidad, por lo que hay que ajustarse a los módulos preestablecidos, que en la mayoría de los casos, no cumplen con todos los requerimientos del cliente.

Ante la problemática descrita anteriormente, y buscando mayor flexibilidad y profesionalidad a la hora de extenderse más allá de las operaciones básicas de administración, con un alto grado de personalización, es necesario la utilización de un *Framework*, que en el campo de la informática, no es más que una estructura conceptual que, mediante librerías, funciones y módulos, nos facilita la tarea de desarrollar un proyecto web. Es una herramienta que permite y facilita la generación de código fuente en una aplicación web y que, generalmente dispone de una comunidad de desarrollo en Internet, que garantizan el acceso a la documentación, así como el soporte futuro. (5)

A diferencia de los *CMS*, los *Frameworks* exigen de más conocimientos de programación, generalmente son los profesionales en informática, los que pueden asumir un proyecto con estas características. Symfony[6], es uno de los *Frameworks* más potentes y con mayor nivel de adaptabilidad. Se encuentra disponible de forma gratuita en Internet, con la documentación y

[6] Creado en SensioLabs, en 2005, por el francés Fabien Potencier.

soporte garantizado, facilitando considerablemente la creación de sistemas con el lenguaje de programación *PHP*[7]. (6)

El Centro Virtual de Información Tecnológica del Azúcar (Cevita) es una aplicación que actualmente se encuentra en su etapa de validación. A pesar de ser desarrollado en Symfony, aún es necesario la intervención de un especialista en programación para la inserción y visualización en el *Frontend* (escenario de los usuarios anónimos), de nuevos recursos externos (enlaces y alimentadores RSS de sitios web de interés) o de nuevos boletines.

Este trabajo tiene como propósito desarrollar, en el Cevita, utilizando el *Framework* Symfony 1.4, dos nuevas prestaciones que potencian su uso y que van más allá de los módulos básicos que este *Framework* genera, aproximando al software, a las características de un *CMS*. La primera permite a los usuarios editores, a través de un típico formulario web, definir diferentes categorías, y dentro de cada una de éstas, los recursos externos que corresponden, y la segunda es un Sistema Autónomo de Publicación de Boletines.

MATERIALES Y MÉTODOS

Como entorno de desarrollo, se instaló sobre el Sistema Operativo Microsoft Windows 10 64 bit la plataforma WAMP (Windows-Apache-MySQL-PHP) que permite simular un servidor web en una computadora local a través de la creación de *Host* Virtuales. En el caso del entorno de producción, y acorde a la fase Beta de la etapa de validación del Cevita, este fue alojado en un servidor de la Intranet Institucional perteneciente al Instituto Cubano de Investigaciones de los Derivados de la Caña de Azúcar (Icidca). Para el trabajo con el código fuente se utilizó la aplicación de código abierto Atom.

[7] Lenguaje de programación de código abierto (PHP: Hypertext Preprocessor, PHP por sus siglas en inglés).

Una vez establecidos los entornos de desarrollo y producción, y con la posibilidad de acceder al sistema desde el navegador web de cada computadora conectada a la red local, se procedió a la creación de los dos módulos personalizados acorde a las necesidades reales del cliente.

Desarrollo del módulo autónomo "Recursos Externos".

Este módulo tiene como objetivo, disponer de un repositorio de enlaces de Internet y alimentadores RSS, organizados en diferentes categorías y aprobados por los editores y expertos del sistema y el instituto respectivamente. Constará de dos escenarios, el primero permitirá la gestión de sus contenidos y estará disponible en el backend para los editores y administradores del sistema, y el segundo mostrará en el frontend, los enlaces clasificados a los usuarios comunes. Los pasos implementados en Symfony 1.4 para cumplir el propósito trazado se muestran a continuación.

Migración de la Base de Datos con Symfony.

Las migraciones de bases de datos con Symfony, son una manera de actualizar de forma segura el esquema de su base de datos. Estas se encargan de crear las nuevas tablas, añadir nuevos campos o eliminar estos. No es aconsejable modificar manualmente la base de datos, pues de seguro este proceder traerá inconsistencias futuras al sistema. Para la implementación de la migración se siguieron los pasos siguientes.

1. Diseño del diagrama Entidad-Relación.

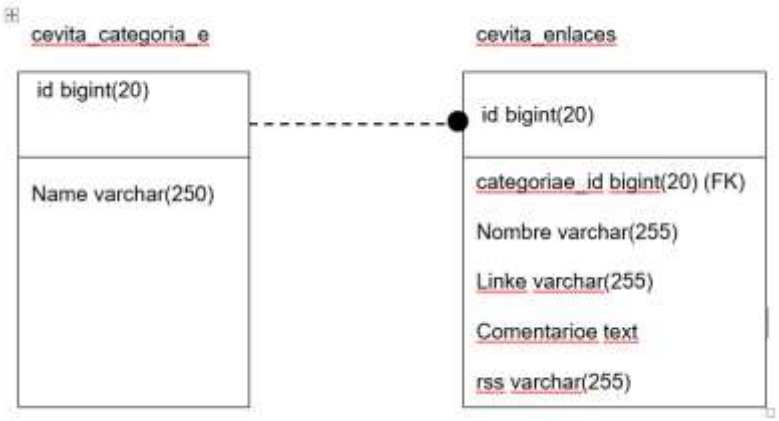

Fig. 1. Diagrama Entidad-Relación. Módulo “Recursos Externos”

En la tabla cevita_categoria_e se almancenarán, como su nombre indica, el nombre de las diferentes categorías en las que se clasifican los recursos externos aprobados para su publicación. Por otro lado la tabla cevita_enlaces contendrá cada uno de los enlaces aprobados por el editor del sistema. Para cada vínculo se podrá almacenar, un nombre que lo indentificará en en Cevita, la URL, un comentario si se requiere sobre el sitio web y por último, la dirección del alimentador RSS en el caso de que disponga.

2. Creando del modelo de datos a partir del diseño anterior.

En este caso crearemos primero el modelo de datos en el archivo shema.yml y luego a partir de este y utilizando las migraciones, crearemos las tablas en la base de datos relacional.

El el caso de Symfony 1.4 el fichero shema.yml se encuentra en la ruta siguiente: Cevita/config/doctrine/. En la Fig. 2 se puede apreciar el modelo de datos correspondiente al diagrama Entidad-Relación de la Fig. 1. Notar en el modelo CevitaEnlaces, la definición de las relaciones existentes, en este caso con el modelo CevitaCategoriaE, incluyendo los detalles relacionados con las claves primarias y foráneas. (Ver Fig. 2)(7)

```
CevitaCategoriaE:
  columns:
    name: {type: string(250), notnull: true}

CevitaEnlaces:
  columns:
    categoriae_id:  {type: integer(8), notnull: true}
    nombre:         {type: string(255), notnull: true}
    linke:          {type: string(255), notnull: true}
    rss:            {type: string(255)}
    comentarioe:    {type: string(500)}
  relations:
    CevitaCategoriaE: {onDelete: RESTRICT, local: categoriae_id, foreign: id, type: one}
```

Fig.2. Extracto del archivo shema.yml. Creando el modelo de datos.

3. Mapeando el modelo a tablas de la base de datos en MySQL a través de las migraciones

Una vez definidos los modelos necesarios en el archivo shema.yml, se procede a ejecutar las migraciones a través de las tareas de Symfony siguientes:

$ php symfony cache:clear

$ php symfony doctrine:generate-migrations-diff

$ php symfony doctrine:migrate

$ php symfony doctrine:build –all-classes

Creando módulo de administración “aenlace” en el backend.

En Symfony es posible generar de forma automática todo el código fuente necesario para ejecutar las tareas de administración (operaciones CRUD) de un módulo determinado. En los casos siguientes, se generan el módulo “autoenlace”, basado en el modelo CevitaEnlaces, para crear las diferentes categorías donde se organizarán los enlaces externos, y el módulo “aenlace”, basado en el modelo CevitaCategoriaE, que permitirá a editores y administradores

del sistema, insertar nuevos recursos externos a la base de datos a través de un formulario web convencional. Todo lo anterior a partir del objeto Doctrine.

$ php symfony doctrine:generate-admin backend CevitaEnlaces -- module=autoenlace

$ php symfony doctrine:generate-admin backend CevitaCategoriaE -- module=aenlace

Creando módulo “Enlaces” para el frontend. Componentes y subconsultas.

Este módulo es creado con el objetivo de mostrar a los usuarios anónimos el listado de recursos externos, aprobados, publicados y clasificados acorde a su categoría. Para este fin se utilizaron los componentes y las subconsultas. Un componente es como una acción, solo que mucho más rápido. La lógica del componente se guarda en una clase que hereda de sfComponents y que se debe guardar en el archivo actions/components.class.php que se encuentra dentro del módulo “Enlaces” del frontend. (Ver Fig. 3)

```
class EnlacesComponents extends sfComponents
    {
      public function executeUenlaces()
      {
        $query = Doctrine::getTable('CevitaEnlaces')
                 ->createQuery()
                 ->groupBy('categoriae_id')
                 ->orderBy('nombre ASC');
                 //->orderBy('fecha DESC')
                 //->limit(10);

        $this->ultenlaces = $query->execute();
      }
    }
```

Fig. 3. Contenido del archivo components.class.php

La presentación del componente se guarda en un archivo que se nombra igual al método que se define en el componente, pero cambiando la palabra execute por un guión bajo y comenzando con minúscula, para este caso sería _uenlaces.php. (Ver Fig. 4)

```
<li>
  <img src="../../Images/bllt__cd_naranja.gif" width="10" height="10">  <b><?php echo $test = $enlace->getCevitaCategoriaE()?></b>
  <?php $test1 = $enlace->getCategoriae_id()?><br>
        <?php
        //subconsulta
        $q = Doctrine_Core::getTable('CevitaEnlaces')
          ->createQuery('c')
          ->where('c.categoriae_id = ?', $test1)
          ->orderBy('c.nombre ASC');

        $y = $q->execute();

          foreach($y as $x) {
         ?>
       <?php if ($x->getComentarioe() == '') {?>

            <a target="_blank" href="http://<?php echo $x->getLinke() ?>" ><?php echo $x->getNombre() ?></a>

                 <?php if ($x->getRss() <> '') {?>
                              <label style="color:blue";>Feed RSS: <?php echo $x->getRss() ?></label><br>
                       <?php }
                      else if ($x->getRss() == ''  ) {?>
                      <br>
                       <?php } ?>
           <?php }
           else  if ($x->getComentarioe() <> '') {?>
          <a target="_blank" href="http://<?php echo $x->getLinke() ?>" title="<?php echo $x->getComentarioe() ?>" >
            <?php echo $x->getNombre() ?></a> <img src="../../Images/cuadro_azul.png" width="10" height="10">

                 <?php if ($x->getRss() <> '') {?>
                            <label style="color:blue";>Feed RSS:  <?php echo $x->getRss() ?></label><br>
                 <?php }
                      else if ($x->getRss() == ''  ) {?>
                        <br>
                  <?php } ?>


           <?php }   ?>
```

Fig.4. Código fuente del archivo Enlaces/templates/_uenlaces.php del *frontend*.

Por último, quedaría incluir el componente en el archivo indexSuccess.php del módulo enlaces para su visualización en la página web.

<?php include_component('Enlaces', 'uenlaces') ?>

Notar en el código anterior, que en los dos primeros atributos corresponden al nombre del módulo donde se definió la acción del componente, en este caso “Enlaces” y seguidamente el nombre de la plantilla de presentación sin el guion bajo del inicio.

Desarrollo del "Sistema Autónomo de Publicación de Boletines"

Este módulo tiene como objetivo disponer de un mecanismo de publicación de boletines creados y/o aprobados por los editores. Constará de cuatro escenarios, el primero permitirá a través de un formulario web, definir el nombre de los diferentes boletines del sistema, en el segundo se podrán realizar todas las operaciones CRUD relacionadas con cada uno de los boletines que se encuentran en la plataforma, estos dos primeros escenarios se encontrarán en el backend, limitando el acceso a estos a solo editores y administradores. La tercera interfaz se visualizará en el frontend para consulta de todos los usuarios comunes, y estará disponible en el menú lateral izquierdo de la página de inicio y bienvenida al centro virtual, en este caso solo se mostrarán los últimos cuatro boletines publicados en cada tipo de boletín. Al último escenario, también del frontend, se accederá desde esta misma casilla lateral, a través del vínculo "Ver más…", y permite entrar al registro de todos los boletines publicados, incluyendo un formulario que, a través de diferentes criterios de búsqueda, brinda la posibilidad de filtrar y consultar el registro de boletines publicados. Los pasos implementados en Symfony 1.4 para cumplir el propósito trazado se muestran a continuación.

Migración de la Base de Datos con Symfony.

Para la implementación de la migración se siguieron los pasos siguientes.

1. Diseño del diagrama Entidad-Relación.

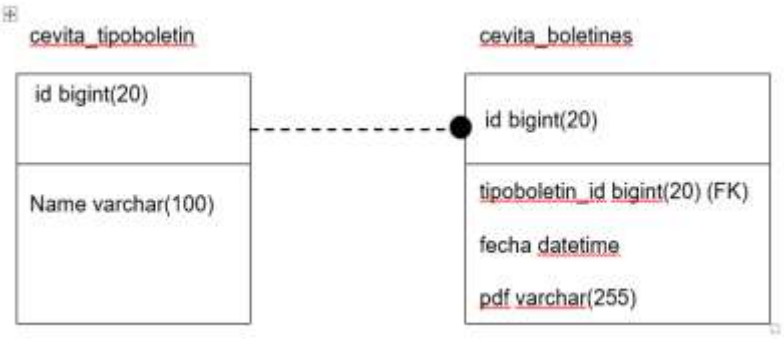

Fig. 5. Diagrama Entidad-Relación. Módulo "Boletines"

En la tabla cevita_tipoboletin se almancenarán, como su nombre indica, el nombre los diferentes boletines que se creen. Por otro lado la tabla cevita_boletines, contendrá cada uno de los boletines aprobados y publicados por el editor del sistema. Para cada boletín se podrá almacenar, el nombre del boletín, la fecha de publicación y el boletín digital en formato pdf.

2. Creando del modelo de datos a partir del diseño anterior.

Como en el ejemplo anterior, crearemos primero el modelo de datos en el archivo shema.yml y luego a partir de este y utilizando las migraciones, crearemos las tablas en la base de datos relacional.

En la Fig. 6 se puede apreciar el modelo de datos correspondiente al diagrama Entidad-Relación de la Fig. 5. Notar en el modelo CevitaTipoboletin la definición de las relaciones existentes, en este caso con el modelo CevitaBoletines, incluyendo los detalles relacionados con las claves primarias y foráneas.(Ver Fig. 6)

```
CevitaTipoboletin:
  columns:
    name: {type: string(100), notnull: true}

CevitaBoletines:
  columns:
    fecha:           {type: timestamp(25), notnull: true}
    tipoboletin_id:  {type: integer(8), notnull: true}
    pdf:             {type: string(255)}
  relations:
    CevitaTipoboletin: {onDelete: RESTRICT, local: tipoboletin_id, foreign: id, type: one}
```

Fig.6. Extracto del archivo shema.yml. Creando el modelo de datos.

3. Mapeando el modelo a tablas de la base de datos en MySQL a través de las migraciones

Una vez definidos los modelos necesarios en el archivo shema.yml, se procede a ejecutar las migraciones a través de las tareas de Symfony siguientes:

$ php symfony cache:clear

$ php symfony doctrine:generate-migrations-diff

$ php symfony doctrine:migrate

$ php symfony doctrine:build –all-classes

Creando módulo “aboletin” de administración en el backend.

En los casos siguientes, se generan el módulo “tipoboletin” basado en el modelo CevitaTipoboletin, para crear los diferentes tipos de boletines, y el módulo “aboletin”, basado en el modelo CevitaBoletines, que permitirá a editores y administradores, insertar nuevos boletines a la base de datos a través de un formulario web convencional. Todo lo anterior a partir del objeto Doctrine.

$ php symfony doctrine:generate-admin backend CevitaTipoboletin -- module=tipoboletin

$ php symfony doctrine:generate-admin backend CevitaBoletines -- module=aboletin

Con el objetivo de subir los boletines en formato pdf al Sistema, es necesario editar el archivo “CevitaBoletinesForm.class.php” que se encuentra en la ruta “lib/form/doctrine”, de forma tal que el *widget*[8] nombrado “pdf” (nombre de la columna de la tabla cevita_boletines y del campo del formulario de entrada de datos) quede como se muestra en la figura 7.

[8] Nombre que otorga Symfony a cada campo de un formulario.

```php
class CevitaBoletinesForm extends BaseCevitaBoletinesForm
{
  public function configure()
  {
    parent::setup();
    $this->setWidget('pdf',    new sfWidgetFormInputFileEditable(
                                    array('label'        => 'Boletín (pdf)',
                                          'file_src'     => $this->getObject()->getPdf(),
                                          'edit_mode'    => !$this->isNew(),
                                          'delete_label' => 'Eliminar',
                                          'template'     => '<div>%file%<br/>%input%<br/>%delete%<br/>%delete_label%</div>'
                                        )));

   //Validadores
   $mime_opt = 'application/pdf';
   $this->setValidator('pdf',    new sfValidatorFile(array('required'   => true,
                                                           'path'       => sfConfig::get('sf_upload_dir').'/digitalesboletines',
                                                           'mime_types' => array($mime_opt),)));
  $this->validatorSchema['pdf']->setMessages(array('required' => 'Campo requerido. Debe adjuntar el boletín.'));
```

Fig. 7. Editando el widget “pdf” para subir boletines en formato digital (pdf) al sistema.

El módulo “content” del frontend. Componentes y subconsultas.

Este módulo es creado con el objetivo de mostrar a los usuarios anónimos el listado de boletines aprobados y publicados. En este particular serán mostrados los cuatro últimos boletines publicados de cada tipo de boletín definido. Esta información será visualizada en la casilla lateral de la página de bienvenida del frontend, nombrada “ÚLTIMOS BOLETINES”.

Para este fin se utilizaron los componentes y las subconsultas. En este caso la lógica se almacena en el archivo actions/components.class.php que se encuentra dentro del módulo “aboletinFront” del frontend.(Ver Fig. 8)

```php
class aboletinFrontComponents extends sfComponents
    {
      public function executeUboletines()
      {
        $query = Doctrine::getTable('CevitaBoletines')
                 ->createQuery()
                 ->groupBy('tipoboletin_id')
                 ->orderBy('id DESC')
                 //->orderBy('fecha DESC')
                 ->limit(20);

        $this->ultboletines = $query->execute();
      }
    }
```

Fig. 8. Contenido del archivo components.class.php

La presentación del componente se guarda en un archivo que se nombra igual al método que se define en el componente, pero cambiando la palabra execute por un guión bajo, y comenzando con minúscula, para este caso sería _uboletines.php. (Ver Fig. 9)

```
<?php foreach($ultboletines as $boletin): ?>
  <li>
    <?php echo $test = $boletin->getCevitaTipoboletin()?>
    <?php $test1 = $boletin->getTipoboletin_id()?><br>
            <?php
            $q = Doctrine_Core::getTable('CevitaBoletines')
              ->createQuery('c')
              ->where('c.tipoboletin_id = ?', $test1)
              ->orderBy('c.fecha DESC')
              ->limit(4);
            $y = $q->execute();
              foreach($y as $x) {
?>
                  <a target="_blank" href="../uploads/digitalesboletines/<?php echo $x->getPdf(); ?>">
             <?php } ?>
  </li>
  <br>
            <?php endforeach ?>
<?php //endif ?>
  </ul>
</div>
```

Fig. 9. Código fuente del archivo aboletinFront/templates/_uboletines.php del frontend.

Para concluir este ejemplo quedaría incluir el componente en el archivo indexSuccess.php del módulo “content” del frontend para su visualización en la página de bienvenida del sistema.

<?php include_component('aboletinFront', 'uboletines') ?>

RESULTADOS Y DISCUSIÓN

El Cevita, con herramientas de la Web 2.0 y Symfony, fue dotado de dos módulos que le aportan un mayor dinamismo, un paso de avance en la construcción del CMS personalizado. En estos momentos los editores del sistema (especialistas en información científica del instituto), no necesitan de un programador para publicar sus boletines, aun cuando se cree un nuevo tipo de boletín.

Por otra parte, la plataforma se enriquece, a una mayor velocidad, con nuevos recursos externos, disponibles en Internet, con un flujo de trabajo mucho más dinámico, donde el experto o investigador puede proponer un enlace de interés, y una vez aprobado este, puede ser publicado rápidamente, con un ahorro considerable de tiempo. Los enlaces pueden ser editados en caso de modificación, o eliminados en el supuesto que el sitio web haya dejado de brindar los servicios, minimizando de esta forma, los posibles vínculos fallidos tan molestos a la hora de buscar información.

Los formularios web creados con Symfony permiten a los editores y administradores, gestionar la información de forma rápida y segura, además de modo automático, ser mostrada en el frontend para los usuarios finales. Notar que como consecuencia del mapeo realizado a través de las migraciones, en cada formulario de entrada, aparece automáticamente, un campo tipo lista, que permite seleccionar las categorías de enlaces y los tipos de boletines. (Ver Fig. 10, 11, 12, 13 y 14).

Fig. 10. Backend. Formulario para la inserción de nuevos enlaces externos.

Fig. 11. Frontend. Listado de enlaces externos según categorías creadas.

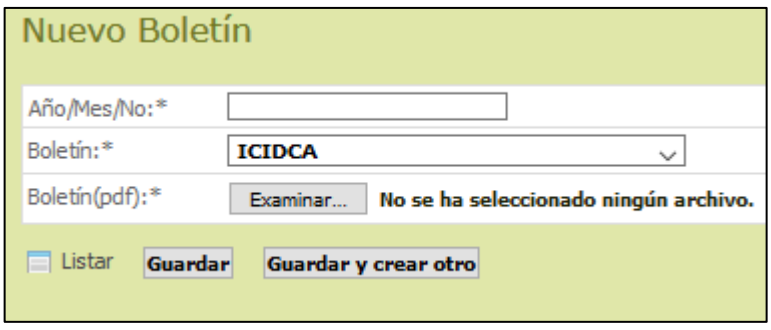

Fig. 12. Backend. Formulario para la inserción de nuevos boletines.

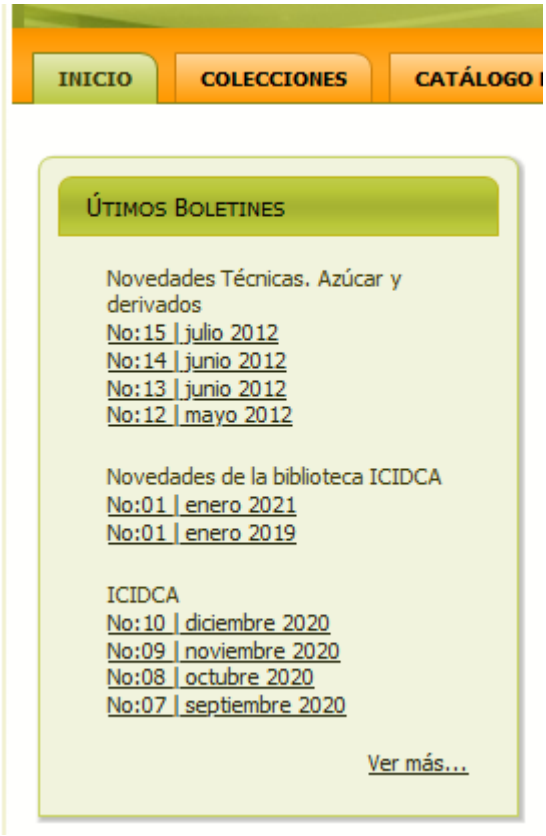

Fig. 13. Frontend. Casilla de contenido lateral. Listado de los últimos cuatro boletines publicados organizados según tipo de boletín. Descargables en pdf.

.

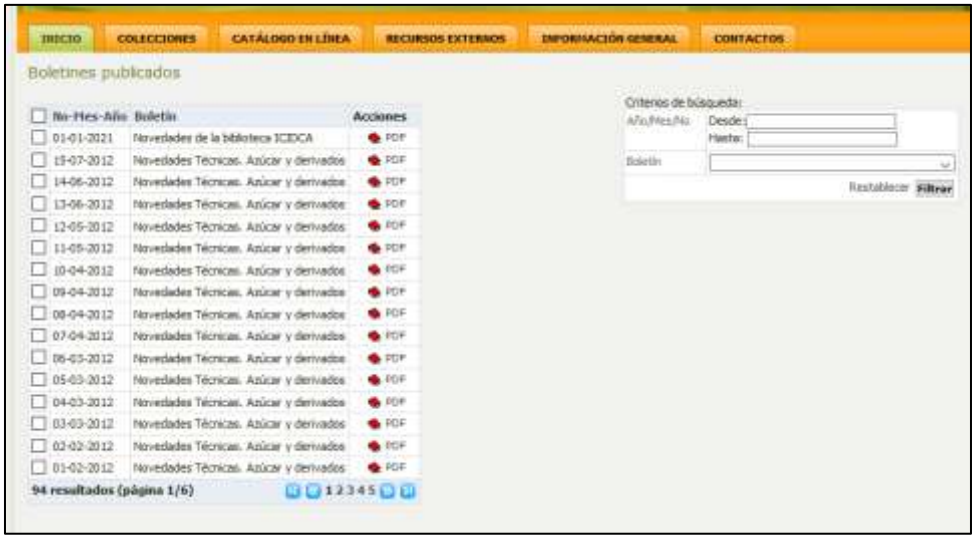

Fig. 14. Frontend. Listado de boletines publicados organizados cronológicamente. Descargables en pdf. Criterios de búsquedas disponibles en el formulario a la derecha.

CONCLUSIONES

La selección de un *CMS* o un *Framework* para el desarrollo de un proyecto web, depende de varios factores como son, el tiempo disponible para el estudio de su implementación y mantenimiento, cuán alejado se encuentran los *CMS* de nuestras necesidades reales, y cuán preparados estamos (conocimiento de programación) para adaptar nuestro sistema a estos requerimientos exigidos, y de esta forma dar respuesta efectiva a problemas y necesidades concretas.

En cualquier caso es importante tener conocimientos avanzados de programación, esto potencia las aplicaciones, dotándolas de mayor profesionalidad, seguridad, adaptabilidad, portabilidad y escalabilidad.

En el ámbito de las investigaciones científicas la implementación de módulos personalizados es una garantía que brinda al investigador la posibilidad de acceso oportuno y confiable a información científica, y no solo de gestionar contenidos e información, sino también de generar conocimientos, principalmente a través de los puntos de intercambio y otros espacios donde el usuario pasa de jugar un rol pasivo a activo, siendo protagonista de los contenidos que publica y nutriéndose de las experiencias de expertos y otros usuarios en general, todo esto, sin necesidad de un intermediario especialista en informática.

Los dos ejemplos desarrollados evidencian la importancia de contar con sistemas autónomos y personalizados de publicación de contenidos, logrando incrementar considerablemente la velocidad de publicación y gestión, así como brindando un acceso confiable y oportuno a la información científica, que una vez pasada a través de un proceso editorial, se dispone de esta, en el momento y lugar que corresponde.

Los resultados alcanzados pueden ser utilizados de referencia a desarrolladores que pretendan incluir en sus aplicaciones características dinámicas de la Web 2.0.

RECOMENDACIONES

Es necesario continuar potenciando el Cevita, incluyendo características dinámicas de la Web 2.0 y acercándolo a la fuerza de un CMS, aprovechando las posibilidades que Symfony ofrece. No solo diseñando nuevos módulos que permitan a los usuarios editores nutrir al sistema de nuevos contenidos, sino, fomentando el desarrollo de lo que se conoce como Contenidos Generados por los Usuarios (*UGC*, por sus siglas en inglés), donde los usuarios comunes, pasan de jugar un rol pasivo a activo, participando activamente en puntos de encuentro o de intercambio de experiencias, y dotándolos de posibilidades para subir contenidos al sistema, que podrían ser posteriormente evaluados, aprobados y publicados por editores y expertos.

AGRADECIMIENTOS

A mi hijo, por el tiempo que no estoy con él.
A mi esposa, por su amor y comprensión.
A mis padres y hermano, por ser quién soy.
A mis Jefes Mauricio y Sabadí, por su exigencia, confianza y guía.
A mis colegas de la Biblioteca del Icidca, por su apoyo incondicional.

REFERENCIAS BIBLIOGRÁFICAS.

1. Barker, D. Web Content Management. Systems, Features and Best Practices. First Edition. USA: O´Relly Media, Inc, 2016. 419.
2. Benevolo, C.; Negri, S. Evaluation of Content Management Systems (CMS): a Supply Analysis. Electronic Journal of Information Systems Evaluation. 9-22. 10, 1 2007.

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.67.3211&rep=rep1&type=pdf [2021-01-25].

3. Ninoriya, S.; Chawan, P. M.; Meshram, B. B. CMS, LMS and LCMS For eLearning. International Journal of Computer Science. 644-647. 8, 2, 3 2011. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.9531&rep=rep1&type=pdf#page=665 [2021-01-23].
4. Patel, S. K.; Rathod, V. R.; Prajapati, J. B. Performance Analysis of Content Management Systems- Joomla, Drupal and WordPress. International Journal of Computer Applications. 39-43. 21, 4, 5 2011. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.206.3027&rep=rep1&type=pdf [2021-01-20].
5. Rodríguez, L. E.; Hernández, A.; Sori, C. Evaluación del Impacto del Uso del Framework Symfony 4 en la Estimación del Esfuerzo de Desarrollo de Software. La Habana. IX Taller Internacional de Calidad en las Tecnologías de la Información y las Comunicaciones (CALIDAD 2020). 16-03-2020. 1-7. <http://www.informaticahabana.cu/sites/default/files/ponencia-2020/CAL38.pdf> [2021-01-22].
6. Bao, J. K. Developing a CMS PHP Framework with Symfony. Thesis for the degree of Master of Science in Software Engineering. California State University, Northridge. 2016.
7. Potencie, F.; Zaninotto, F. Symfony la guía definitiva. 2008.<http://www.librosweb.es> [2020-12-16].