

# Data

## aplicacion de gestion de datos de socios, reservas y acreditaciones

El presente documento describe la aplicación llamada Data; creada para gestionar la información de datos de Socios, Reservas y Acreditaciones relacionadas con la Convención CifiMad.

## Accesos y ubicaciones

El acceso a dicha aplicación se halla en <http://data.cifimad.es>

Existe una versión de pruebas ubicada en <http://datatest.cifimad.es> (no actualizada – solicitar actualización antes de empezar a probar, o subir código y cambios en base de datos)

El repositorio de código fuente se ubica en <https://github.com/mvillarl/cifimad-data>

La aplicación está alojada en el proveedor Strato. Se puede acceder a la ubicación de los ficheros de código a través de la conexión FTP:

Servidor: [ftp.strato.com](ftp://ftp.strato.com)

Usuario: [ftp\_tienda@cochranemadrid.es](mailto:ftp_tienda@cochranemadrid.es)

Contraseña: Cochrane1428

Directorio: /data para la versión real, /datatest para la de pruebas

La base de datos asociada es:

Servidor: rdbms.strato.de

Usuario: U2627984 para la versión real, U1438945 para la de pruebas

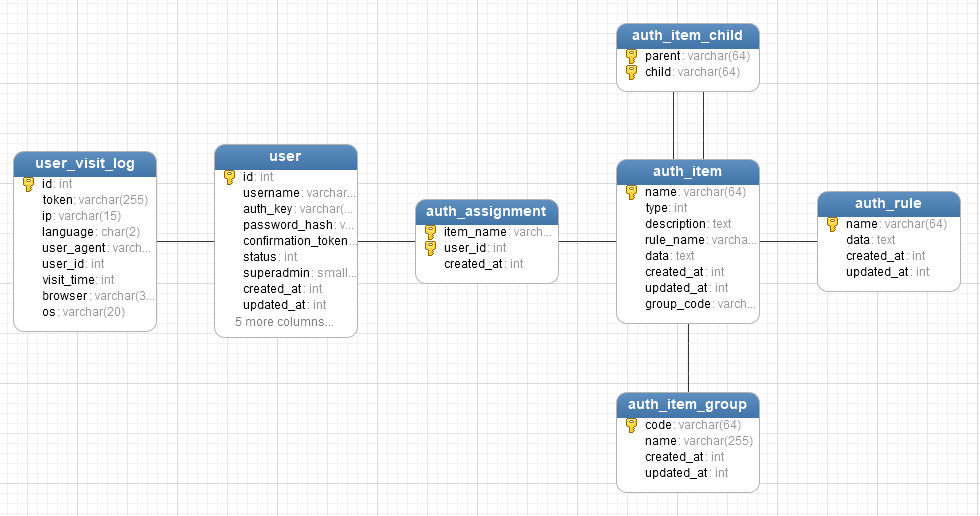
Contraseña: vR3ntH5pFn

Base de datos: DB2627984 para la versión real, DB1438945 para la de pruebas

## Modelo de datos

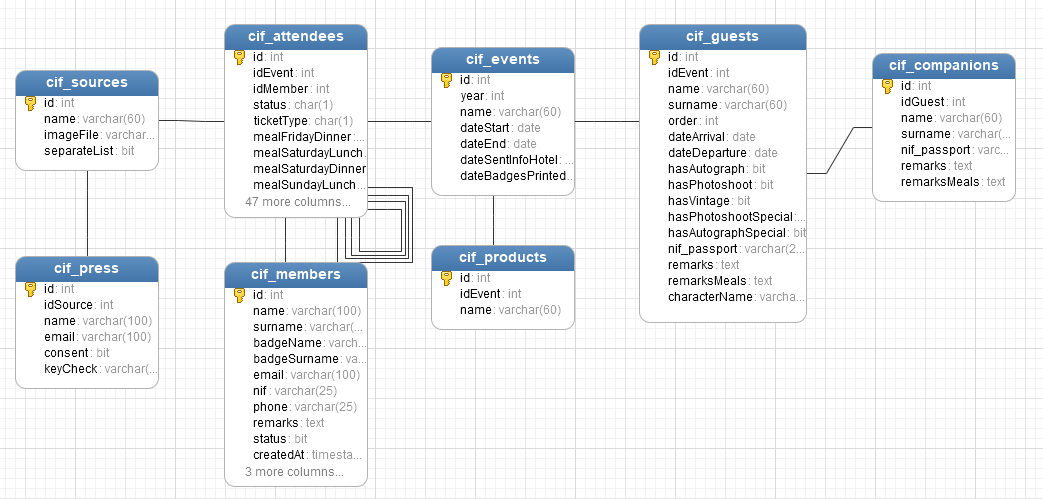
### Usuarios

Este diagrama se corresponde con el modelo de datos de usuarios y permisos. No se utiliza directamente, sino que lo gestiona el módulo Webvimark User Management, pero se incluye como referencia.



### CifiMad

Este sí es el modelo de datos que va a tratar la aplicación directamente, por lo que lo describiremos en detalle.



#### Tabla cif\_events – Eventos

Esta tabla recoge la información de cada evento CifiMad.

* Id – identificador del evento
* Year – año en el que tiene lugar
* Name – nombre del evento (normalmente “CifiMad <año>”)
* dateStart – fecha de inicio
* dateEnd – fecha de fin
* dateSentInfoHotel – fecha y hora en la que se envía la información de reservas al hotel
* dateBadgesPrinted – fecha y hora en la que se imprimen las acreditaciones

#### Tabla cif\_products – Productos extra

En esta tabla encontramos los productos que se salen del patrón habitual de cenas / firmas / fotos / cartones: cartones no pertenecientes a un invitado, cartones adicionales al principal, entradas de torneos, etc.

* Id – identificador del producto
* idEvent – clave foránea: identificador del evento
* name – nombre del producto

#### Tabla cif\_guests – Invitados

En esta tabla manejamos la información de los invitados estrella del evento.

* Id – identificador del invitado
* Name – nombre
* Surname – apellido
* Order – orden en el cartel
* dateArrival – fecha de llegada al hotel
* dateDeparture – fecha de salida del hotel
* hasAutograph – indica si se venden autógrafos de este invitado
* hasPhotoshoot - indica si se venden fotos profesionales con los asistentes para este invitado
* hasVintage – indica si se vende cartón vintage del personaje más conocido de este invitado
* hasAutographSpecial - indica si se venden autógrafos especiales de este invitado
* hasPhotoshootSpecial - indica si se venden fotos profesionales especiales con los asistentes para este invitado
* nif\_passport – nº de DNI o pasaporte del invitado
* remarks – observaciones
* remarksMeals – observaciones referentes a comidas y cenas
* characterName – nombre del personaje más conocido de este invitado
* pseudonym – seudónimo para reserva de hotel

#### Tabla cif\_companions – acompañantes

Esta tabla recoge la información sobre los acompañantes de los invitados

* Id – identificador del acompañante
* IdGuest – clave foránea; identificador del invitado al que acompaña
* Name – nombre
* Surname – apellido
* badgeName – nombre en la acreditación
* badgeSurname – apellido en la acreditación
* nif\_passport – nº de DNI o pasaporte del invitado
* remarks – observaciones
* remarksMeals – observaciones referentes a comidas y cenas
* separateRoom – indica si se aloja en habitación separada
* excludeLodging – indica si no se aloja
* excludeFridayDinner – indica si no se queda a cenar el viernes

#### Tabla cif\_sources – procedencias

Esta tabla maneja la información de las procedencias de los asistentes

* id – identificador de procedencia
* name – nombre
* imageFile – nombre del fichero de imagen (para incluir el logo en la acreditación)
* separateList – indica si la acreditación va en lista separada (usado para asociaciones grandes)
* blankBadges – indica si se deben imprimir acreditaciones en blanco (con logo) para esta procedencia

#### Tabla cif\_members – socios

Esta tabla gestiona los datos de los “socios” de CifiMad, es decir, cualquier persona que asista o haya asistido a alguno de los eventos.

* Id – identificador de socio
* Name – nombre
* Surname – apellidos
* badgeName – nombre en la acreditación
* badgeSurname – apellido en la acreditación
* email – correo electrónico
* nif – nº NIF
* phone – número de móvil
* remarks – observaciones
* status – estado (activo o inactivo)
* createdAt – fecha y hora de creación
* updatedAt – fecha y hora de última modificación
* consent – indica si el socio ha dado su consentimiento para recibir mails
* keyCheck – clave de verificación usada para recoger el consentimiento
* small – indica si se debe mostrar la etiqueta de acreditación con letra más pequeña

#### Tabla cif\_attendees – asistentes

En esta tabla recogemos los datos de los asistentes a uno de los eventos.

* Id – identificador de asistente
* idEvent – clave foránea: identificador del evento
* idMember – clave foránea: identificador del socio
* status – estado del asistente (pendiente de pago, confirmado, cancelado)
* ticketType – tipo de acreditación (día o fin de semana)
* mealFridayDinner – indica si ha adquirido cena para el viernes
* mealSaturdayLunch – indica si ha adquirido comida para el sábado
* mealSaturdayDinner – indica si ha adquirido cena para el sábado
* mealSundayLunch – indica si ha adquirido comida para el domingo
* mealSundayDinner – indica si ha adquirido cena para el domingo
* guest1Photoshoot – fotos adquiridas con el invitado 1
* guest1PhotoshootSpecial – fotos especiales adquiridas con el invitado 1
* guest1Autograph – firmas adquiridas del invitado 1
* guest1AutographSpecial – firmas especiales adquiridas del invitado 1
* guest1Vintage – cartones vintage adquiridos del invitado 1
* (campos repetidos hasta guest4)
* extraProduct1 – unidades adquiridas del producto extra 1
* (campo repetido hasta extraProduct4)
* idSource – clave foránea: identificador de procedencia
* isSpecial – indica si es una acreditación especial (doble y en letras rojas, para miembros del Cochrane)
* roomType – tipo de habitación reservada
* dateStartLodging – fecha de inicio de alojamiento
* dateEndLodging – fecha de fin de alojamiento
* idAttendeeRoommate1 – clave foránea: identificador del asistente que comparte habitación
* idAttendeeRoommate2 – clave foránea: identificador del asistente que comparte habitación
* idAttendeeRoommate3 – clave foránea: identificador del asistente que comparte habitación
* freeLodging – invitado a alojamiento (en desuso)
* freeSaturdayDinner – invitado a cena del sábado (en desuso)
* remarks – observaciones internas
* remarksRegistration – observaciones de acreditaciones
* remarksMeals – observaciones sobre comidas – general
* remarksMealSaturday – observaciones sobre comidas – cena del sábado
* remarksHotel – observaciones sobre la reserva de hotel
* orders – identificador de pedido o pedidos (separados por comas) en la tienda online
* createdAt – fecha y hora de alta
* updatedAt – fecha y hora de última modificación
* updatedAtHotel – fecha y hora de última modificación de datos de reserva de hotel
* updatedAtBadges – fecha y hora de última modificación de datos de acreditación
* updatedAtBadgesTickets – fecha y hora de última modificación de datos de acreditación y compras
* idAttendeeParent – clave foránea; si el asistente es un niño, identificador del asistente padre / madre / acompañante adulto

#### Tabla cif\_press – medios de prensa

En esta tabla guardamos los medios de prensa a los que queremos enviar comunicados

* id – identificador de medio de prensa
* idSource – clave foránea; identificador de procedencia asociada al medio
* name – nombre del medio
* email – correo electrónico del medio
* consent – indica si el socio ha dado su consentimiento para recibir mails
* keyCheck – clave de verificación usada para recoger el consentimiento

## Aplicacion

La aplicación está construida sobre el framework Yii versión 2.0.8; a partir de los controladores, modelos y vistas generados por Gii (el módulo de creación de aplicaciones de Yii a partir de tablas de base de datos). En algunos casos muy modificadas.

Este documento describe las modificaciones hechas, puesto que ya existe documentación abundante sobre las funcionalidades de Yii y sus aplicaciones generadas con las opciones por defecto.

### Sistema de usuarios y permisos

Como se menciona en el apartado de modelo de datos, se ha utilizado un módulo de extensión disponible libremente en internet para añadir esta funcionalidad a la aplicación. Se puede gestionar con los permisos adecuados (actualmente, sólo el usuario superadmin los tiene), en la sección Gestión de usuarios.

#### Usuarios

Además de nombre, correo electrónico, contraseña y estado, a un usuario se le puede asignar uno o más roles.

#### Roles

Un rol es un conjunto de permisos.

#### Permisos

Cada permiso representa un conjunto de accesos a las diferentes partes de la aplicación, definidas como acciones de controladores. Dado que Gii genera por defecto un controlador para cada objeto, con una acción para cada funcionalidad CRUD (alta, acceso, modificación y borrado), nos aparecerán estas acciones para asignar permiso. Además, nos pueden aparecer las nuevas acciones de controladores que creemos.

Si venimos a este apartado después de haber modificado un controlador, debemos refrescar las rutas para que nos aparezcan como opciones las nuevas acciones que hayamos dado de alta.

Los permisos se pueden organizar en grupos, para que aparezcan ordenados y más fáciles de leer a la hora de asignárselos a roles o a usuarios.

### .htaccess / Virtual host

Para el funcionamiento correcto de la aplicación es necesario definir un host virtual con determinados parámetros:

* El directorio raíz no debe ser la raíz de la instalación, sino el directorio web dentro de ésta.
* Para que funcionen los URLs “amigables”, se debe usar mod\_rewrite para redirigir todas las rutas a index.php

Un ejemplo (si el proyecto se ha instalado en C:\webs\data.cifimad.es):

<VirtualHost \*:80>

ServerName data.cifimad.local

AddType application/x-httpd-php .html .php

DirectoryIndex index.html index.htm index.php index.php3 index.cgi

DocumentRoot "C:/webs/data.cifimad.es/web"

ErrorLog "data.cifimad.es\_error.log"

<Directory "C:/webs/data.cifimad.es/web">

Options Indexes FollowSymLinks MultiViews ExecCGI Includes

AuthType none

AllowOverride All

Order allow,deny

Allow from all

Require local

</Directory>

<Directory "C:/webs/data.cifimad.es/web">

<IfModule mod\_rewrite.c>

RewriteEngine on

# prevent httpd from serving dotfiles (.htaccess, .svn, .git, etc.)

RedirectMatch 403 /\..\*$

# if a directory or a file exists, use it directly

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

# otherwise forward it to index.php

RewriteRule .\* index.php

</IfModule>

</Directory>

</VirtualHost>

Como se ve, está asociado al nombre data.cifimad.local; se deberá definir en /etc/hosts apuntando a 127.0.0.1

### Instalar paquetes

Para el correcto funcionamiento de la aplicación es necesario actualizar los paquetes externos, que no se incluyen en la base de código en Git: tanto el propio Yii como algunas extensiones adicionales. Para ello basta con ejecutar en el directorio de instalación el comando:

composer update

Composer es la forma recomendada de empezar a trabajar con Yii, así que ya debería estar presente en el sistema. De no ser así, se explica cómo obtenerlo en:

https://getcomposer.org/download/

### Configuración

El punto de entrada a la aplicación index.php está modificado para definir las variables $\_prod y $\_ppe, que indican si estamos en el entorno real (“producción”) o de pruebas (“pre-production environment”) respectivamente.

Las configuraciones que se han definido en el directorio config estándar de Yii son:

#### web.php

Hay una serie de configuraciones específicas de la aplicación:

* Los valores estándar id y name son diferentes según el entorno.
* components / user y modules / user-management hacen que se cargue el módulo de autenticación y permisos.
* Además del objeto estándar de configuración de conexión a la base de datos db, hay otro llamado dbwordpress (ver más adelante su sección específica), y uno en desuso llamado dbtienda (que conectaba a la base de datos de la tienda Prestashop en uso hasta 2017).
* components / urlmanager define una serie de rutas necesarias para poder pasar parámetros a determinadas acciones de controladores. Las describiremos en detalle al hablar de los objetos Member y Attendee.
* defaultRoute define el listado de asistentes como el punto de entrada a la aplicación.
* on beforeAction es el manejador de eventos que se ejecuta previamente a todas las acciones. Lo usamos para obligar al usuario a cambiar la contraseña si así lo ha definido el administrador; y para cambiar la vista usada como layout cuando se accede a determinadas acciones de determinados controladores:
  + publicLayout para las páginas que tiene que ver el público (con el diseño de la web CifiMad): los consentimientos de socios y medios de prensa
  + excelLayout para exportaciones a (pseudo-)Excel
  + reportLayout para informes
  + ningún layout para consultas que devuelven datos en formato Ajax
* modules / user-management tiene su propio manejador on beforeAction. Aquí aplicamos el layout especial para la página de login, y repetimos la funcionalidad de obligar a cambiar contraseña.

#### db.php

Está un poco modificado respecto de la configuración estándar, porque cambia según el entorno:

* La dirección del servidor es diferente cuando no estamos ni en el entorno real ni en el de pruebas (se entiende entonces que es un entorno de desarrollo)
* La base de datos y el usuario son diferentes en el entorno de pruebas

#### dbwordpress.php

Como describiremos al hablar del objeto Member, existe una segunda conexión a base de datos. Corresponde a la aplicación WordPress / WooCommerce donde reside la web y la tienda online de CifiMad.

### Layout principal

Está en views/layouts/main.php, siguiendo el estándar Yii.

Todos los enlaces sólo se añaden si el método User::canRoute() devuelve true; así nos aseguramos de que un usuario ni siquiera vea una sección a la que no tiene acceso.

Los objetos base están agrupados en la sección Base; los diferentes informes, en la sección Asistentes.

### Objetos de aplicación – Event

Lógicamente está asociado a la tabla cif\_events.

Dado que algunas funcionalidades abarcan varios ficheros, en vez de describir las modificaciones a cada fichero vamos a describir cada funcionalidad y señalar los puntos del código donde se implementan.

#### Lista de valores para el campo Año

La clase modelo models/Event.php define el método getYears(), que devuelve un array de años desde 2016 (año de primera puesta en marcha de la aplicación) hasta un año después del actual. Así, siempre podemos dar de alta el evento para el año que viene.

En la vista de alta / modificación de registro views/event/\_form.php , los valores para el campo año se definen a través del método getYears().

#### Intervalo de fechas

Esta funcionalidad verifica que la fecha de inicio sea realmente menor que la fecha de fin.

En la vista de alta / modificación de registro views/event/\_form.php , se registra el módulo JavaScript datepickerinterval.js, que implementa esta funcionalidad de verificación; y se añaden a los campos relevantes una clase CSS y un atributo personalizado que activan dicha funcionalidad. Además, los campos usan el widget DatePicker.

#### Campos de fecha y hora de impresión de acreditaciones y de envío de datos al hotel

El uso que se le va a dar a estos campos consiste en, en determinados momentos, asignarle la fecha y hora actual. Por tanto, en vez de dejarlos abiertos y obligar al usuario a introducirla manualmente, los definimos como sólo lectura y añadimos unas casillas de verificación; sólo si el usuario las marca, al guardar los cambios se asigna al campo el valor correspondiente a la fecha y hora actual.

La clase modelo models/Event.php sobreescribe el método load() (que el framework llama para asignar valores a los campos del modelo a partir del formulario de alta / modificación); en este punto se leen los campos correspondientes a las casillas de verificación, y si están marcadas, se le da valor a los campos correspondientes a la fecha y hora.

En la vista de alta / modificación de registro views/event/\_form.php , los campos de fecha y hora de impresión de acreditaciones y envío al hotel están definidos como sólo lectura, y se les añade un texto de ayuda; y justo debajo de cada uno, se añade una casilla de verificación.

#### Evento siguiente, último evento

Estos métodos leen el último evento de todos los que haya almacenados, y el siguiente a partir de la fecha actual. Los usa el módulo de asistentes para asignar el evento que se va a usar por defecto.

### Objetos de aplicación – productos extra

Asociado a tabla cif\_products

#### Desplegable de eventos

Los valores del campo idEvent son un desplegable que muestra los eventos dados de alta en el sistema; tanto en el alta y modificación de registro, como en el listado para permitir filtrar por evento.

La clase modelo models/Product.php implementa el método getEvents(), que a su vez invoca a la clase modelo models/Events.php para leer los datos necesarios y devolverlos en forma de array bidimensional.

La clase controlador controllers/ProductController.php le pasa esta información de eventos a las vistas correspondientes en las acciones actionIndex(), actionCreate() y actionUpdate().

La vista de listado views/product/index.php incluye los datos de eventos como valores del filtro para la columna idEvent.

La vista views/product/\_form.php incluye los valores de eventos como valores posibles del campo idEvent.

#### Mostrar el nombre del evento en la información del producto

El controlador controllers/ProductController sobreescribe el método findModel(), que localiza los datos de un registro a partir de su id, para incluir el nombre del evento en la consulta a la base de datos. Le asigna el nombre de campo eventName.

La clase modelo models/ProductSearch.php modifica de forma análoga la consulta a base de datos del método search()

El modelo models/Product.php define la propiedad eventName.

La propiedad eventName se muestra en las vistas de listado views/product/index.php y visualización de registro views/product/view.php

### Objetos de aplicación – invitados

Asociado a tabla cif\_guests

#### Desplegable de eventos

La funcionalidad es idéntica a la descrita para los productos extra.

#### Mostrar el nombre del evento en la información del producto

La funcionalidad es idéntica a la descrita para los productos extra.

#### Nombre completo

La clase modelo models/Guest.php define el método getFullname() como concatenación de name y surname. Eso hace que el sistema reconozca fullname como una propiedad más del modelo Guest. Esta propiedad se usa a lo largo de la aplicación para mostrar datos del invitado.

#### Intervalo de fechas

La misma funcionalidad descrita para el objeto Evento, se aplica en el objeto Invitado para los campos dateArrival y dateDeparture.

### Objetos de aplicación – acompañantes

Asociado a tabla cif\_companions

#### Desplegable de invitados

Esta funcionalidad es análoga al desplegable de eventos descrito para los objetos Producto extra e Invitado, pero con valores de invitados en vez de eventos, asociados al campo idGuest. Sólo se aplica al alta y modificación, en este caso no se ha visto necesidad de añadirlo como filtro en el listado.

Los invitados están filtrados por evento; el valor del evento se toma de la sesión, donde previamente se ha almacenado por el controlador de Asistentes (ver más adelante).

#### Nombre completo

Funcionalidad análoga a la descrita para Invitado.

### Objetos de aplicación – procedencias

Asociado a tabla cif\_sources

#### Imagen

Uno de los campos de la tabla representa una imagen; esto quiere decir que, además de guardar el nombre de la imagen en la base de datos, hay que gestionar el fichero físico.

La clase controlador controllers/SourceController.php, en las acciones actionCreate() y actionUpdate(), utiliza el componente estándar de Yii UploadedFile para obtener los datos del fichero que el usuario ha subido desde su equipo, lo asocia al modelo, e invoca el método upload() de éste.

El modelo define la propiedad imageFileObj que será un objeto UploadedFile; y el método upload() que valida los datos recibidos del fichero y, si son correctos, los almacena en el directorio correspondiente en el disco y además asigna el nombre en la propiedad imageFile.

La vista de registro views/source/view.php muestra la imagen si el campo imageFile tiene contenido.

La vista de alta / modificación views/source/\_form.php muestra la imagen si el campo imageFile tiene contenido, y además, un campo de tipo fichero para subir una imagen nueva o sustituir la existente.

#### Carga masiva

La aplicación incluye una funcionalidad para cargar varias procedencias en un solo paso. Se toma cada línea de la caja de texto como el nombre de una de ellas.

En la vista de listado views/source/index.php se ha añadido un botón para dar acceso a la funcionalidad. Está enlazado con la ruta que lleva a la nueva acción de controlador load.

La clase controlador controllers/SourceController.php implementa el método actionLoad() que implementa la funcionalidad. Enlaza con la nueva vista views/source/load.php

Dicha vista muestra el formulario con la caja de texto, y en su caso el mensaje de error.

### Objetos de aplicación – socios

Asociado a tabla cif\_members

#### Carga masiva

La sección de socios tiene una función de carga masiva, similar a las procedencias pero algo más compleja, por dos elementos:

* Se cargan varios campos (separados por tabuladores, lo que hace que se puedan procesar los datos copiados y pegados desde Excel).
* Los socios cargados se pueden dar de alta como asistentes en el mismo paso

La clase controladora controllers/MemberController.php implementa el método actionLoad() que controla la lógica del proceso. Recoge los datos, inserta los socios y los asistentes en su caso, y le pasa los datos a la plantilla; incluyendo los contenidos de los desplegables.

La clase modelo de asistentes models/Attendee.php implementa getEvents(), getSources() y getTicketTypes() para generar los datos de los desplegables de evento, procedencia y tipo de entrada respectivamente. Los dos primeros leen la información del modelo correspondiente, el tercero es una serie de valores fijos.

La vista views/member/load.php muestra el formulario que recoge los datos; incluyendo los datos para crear asistentes asociados a un evento, con una procedencia y un tipo de entrada determinados.

#### Carga desde la tienda

Es posible cargar socios a partir de los usuarios registrados que han hecho pedidos en la tienda online.

En la vista de listado views/member/index.php hay un botón para acceder a esta funcionalidad.

En el fichero de configuración config/web.php se define la ruta '/member/loadfromps/<filter>' que permite pasar a la acción del controlador un parámetro filter.

La clase controladora controllers/MemberController.php define el método actionLoadfromwp() que implementa el proceso de leer los datos de usuarios de WordPress, verificar los que ya existen en la base de datos y los que no, e insertar/modificar todos o sólo insertar los nuevos según indique el parámetro $filter.

La clase modelo models/Member.php implementa:

* getLastLoadFrom() recupera la fecha y hora de la última vez que se han cargado datos de socios (para saber cuáles son los nuevos). La fecha se guarda en un fichero de texto.
* setLastLoadFrom() asigna la fecha y hora actual como momento de la última carga.
* membersMatchCustomers() recupera los socios existentes cuyo email coincide con los de los clientes cargados desde la tienda.
* upsertMembersFromCustomers() construye objetos Member a partir de los datos de clientes, y los inserta o modifica.

La vista views/member/loadfromps.php muestra los botones para lanzar la carga, y los resultados de la operación.

Existe un modelo models/WPUser.php y su clase de consulta a base de datos asociada models/WPUserQuery.php que se corresponden con la tabla estándar de WordPress wp\_users. WPUser.php define ciertos métodos que devuelven campos ya adaptados para corresponder con los de cif\_members; y redefine el método getDb() para conectarse a la base de datos de WordPress en vez de la de la aplicación Data. WPUserQuery.php añade a los datos de wp\_users los almacenados en la tabla auxiliar wp\_usermeta; y define el método modifiedAfter() que devuelve la consulta a base de datos correspondiente para obtener los usuarios creados o modificados después de una fecha y hora indicada.

Existen clases equivalentes PS2Customer.php y PS2CustomerQuery.php para acceder a la tienda online implementada en Prestashop (en desuso desde 2017).

#### Exportar datos de socios

La aplicación permite exportar los datos de los socios. Esto tiene dos objetivos principales:

* Presentar una relación de los socios de la Asociación CifiMad (entendiendo por socios, todas las personas que hayan asistido a alguno de los eventos y de las que tengamos sus datos, incluyendo DNI).
* Realizar acciones de envío masivo de correo electrónico.

En la vista de listado views/member/index.php hay botones para lanzar estas exportaciones.

Por tanto se ofrecen opciones para exportar todo, exportar sólo los socios activos con DNI, y exportar sólo los socios activos con correo electrónico.

En el fichero de configuración config/web.php se define la ruta '/member/export/<onlydni>' que permite pasar a la acción del controlador el parámetro que indica el filtro a aplicar.

La clase controladora controllers/MemberController.php define el método actionExport() que aplica el filtro indicado, obtiene los datos, establece las cabeceras necesarias para que el navegador no muestre la salida sino que permita descargarla, y le pasa la información a la vista.

La vista views/member/export.php define una sencilla tabla; y, como hemos descrito en la sección de layouts, está asociada a un layout especial que elimina la navegación. Lo que se devuelve no es realmente un Excel sino una tabla HTML, pero se le indican al navegador cabeceras para descargarlo con formato y extensión de fichero Excel; al abrirlo, Excel avisa de que el formato es incorrecto pero lo convierte y a partir de ahí se puede manejar como un documento Excel real.

#### Búsqueda Ajax

Como parte del campo con autocompletado que describiremos en el objeto Asistente, es necesaria una funcionalidad que reciba un término de búsqueda y devuelva una estructura de datos JSON con identificadores y nombres de socios que coincidan.

En el fichero de configuración config/web.php se define la ruta '/member/ajaxsearch/<term>' para poder pasarle el parámetro a la acción del controlador.

El controlador controllers/MemberController.php define la acción actionAjaxsearch ($term), que ejecuta la búsqueda mediante el método memberSearch() del modelo models/Member.php, y devuelve los datos en Json. Como ya describimos, esta acción tiene un layout vacío para que el JSON se devuelva sin ningún HTML alrededor.

#### Fecha de creación y modificación

Como hemos descrito en el modelo de datos, el campo createdAt debe almacenar la fecha y hora de creación del socio, y updatedAt, la de última modificación.

La clase modelo models/Member.php implementa esta funcionalidad mediante el componente de Yii llamado TimestampBehavior.

En la vista de modificación de registro views/member/\_form.php , los dos campos se muestran como de sólo lectura.

#### Nombre completo

El modelo models/Member.php tiene una propiedad FullName exactamente igual a la descrita para invitados y acompañantes; y además, otra similar para el nombre de acreditación llamada FullBadgeName.

#### Gestionar consentimiento

Debido a la nueva ley RGPD, se necesita recabar el consentimiento expreso de los usuarios para poder enviarles correos electrónicos. Lo obtendremos de dos maneras:

* Si un usuario se da de alta en la tienda online, o se identifica y hace un pedido, tiene que dar consentimiento
* Haremos un emailing masivo en el que solicitaremos consentimiento para las siguientes comunicaciones, proporcionando un enlace que al pincharlo recogerá el consentimiento del usuario

En el primer caso, los usuarios que se cargan de la tienda se guardan con el consentimiento dado directamente. Clase modelo app\models\Member.php, método upsertMembersFromCustomers

Además, se verifica qué usuarios hayan dado consentimiento al realizar un pedido, y para los que no tengamos dicho consentimiento actualizado; y permite actualizarlos. En la clase controladora app\controllers\MemberController.php, método \_loadfromshop se verifica la información, y se muestra en la vista views/member/loadfromps; desde dicha vista se invoca la acción definida en la clase controladora app\controllers\MemberController.php, método actionUpdateconsents. En la clase modelo app\models\Member.php, método membersMatchConsents, se ejecuta la lógica de la comparación entre los consentimientos de la tienda y de la base de datos Data. La clase WPUserQuery define el método usersWithConsent() para obtener los usuarios que hayan dado consentimiento al realizar un pedido.

La funcionalidad ya descrita de exportar socios que tengan e-mail, filtra además los que tengan consentimiento. Además se define un botón nuevo (accesible desde views/member/index, que invoca la acción existente actionExport con un parámetro nuevo) para exportar los que tengan e-mail y *no* tengan consentimiento; a esos será a los que se les envíe el emailing masivo. Por tanto, a dicha exportación se le añade la clave de verificación, cuyo funcionamiento se define en el siguiente apartado.

#### Recoger consentimientos

La idea es que, a todos los socios que no hayan dado consentimiento, se les envía un e-mail solicitándolo. Dicho e-mail contendrá un enlace al módulo que se va a describir, que incluye el correo electrónico y una clave de verificación; siempre que se localice un socio para el que coincidan esos dos datos, se actualizará el consentimiento y mostrará un mensaje de verificación.

El campo keyCheck es de sólo lectura en el alta o modificación de socio (vista views/member/\_form). Cuando se da de alta un socio, ya sea manualmente o al cargarlo desde la tienda, se le asigna un keyCheck único y aleatorio (método setKey de la clase modelo app\models\Member ).

Se define una acción en la clase controladora app\controllers\MemberController, método actionSetKeys, para asignar claves de verificación a todos los socios que no lo tengan. No está enlazada puesto que basta con invocarla una vez manualmente.

La acción actionConsent de la clase controladora app\controllers\MemberController es a la que enlazará el correo que se envíe a los socios. No requiere autenticación (al estar indicada en el método getFreeAccessActions() de la clase controladora) y tiene un layout distinto para parecer parte de la web de CifiMad. Recibe el correo y la clave de verificación; si encuentra un socio para el que coincidan los dos valores, actualiza su consentimiento. La vista definida en views/member/consent muestra un mensaje de confirmación si el consentimiento se ha actualizado, o un mensaje de error si el socio no se encuentra.

### Objetos de aplicación – asistente

Asociado a la tabla cif\_attendees

#### Filtro de evento por defecto

No tiene mucho sentido que nada más entrar a la aplicación aparezcan los asistentes de todos los eventos, o del primero; además, dado que los campos de un registro de asistente cambian según el evento, es importante que haya uno seleccionado antes de entrar. Para conseguir esto, la aplicación guarda cuál es el que se ha seleccionado en el filtro del listado, y además, nada más entrar selecciona el del próximo evento, o si no hay ninguno, el del último hasta ahora.

La clase controladora controllers/AttendeeController.php define el método getCurrentEvent(). Dicho método recupera el identificador de evento de la sesión; si no, devuelve el próximo evento, y si no el último (utilizando los métodos del modelo models/Event.php ya descritos).

Todas las acciones recuperan el identificador de evento utilizando este método. actionIndex(), además, lo guarda en sesión.

El método search() de la clase modelo de búsqueda models/AttendeeSearch.php guarda el identificador de evento en la sesión cada vez que se selecciona uno en el filtro de búsqueda.

#### Valores de desplegables en filtros de búsqueda y campos de formulario

Hay una serie de campos definidos como desplegables con una serie de valores fijos posibles:

* Tipos de entrada: valores fijos
* Estado: valores fijos
* Eventos: ids de la tabla cif\_events
* Procedencias: ids de la tabla cif\_sources
* Tipos de habitación: valores fijos

La clase modelo models/Attendee.php define una serie de métodos que devuelven las listas de valores correspondientes:

* getStatusMap()
* getTicketTypes()
* getEvents()
* getSources()
* getRoomTypes()

La clase controlador controllers/AttendeeController.php invoca estos métodos y le pasa las listas de valores correspondientes a las vistas en las acciones actionIndex(), actionCreate() y actionUpdate().

#### Rejilla de opciones en el listado

El listado muestra unas columnas especiales: una rejilla de comidas, y otra de fotos y firmas. En cada una de ellas aparece una columna por cada opción, y un aspa roja o tick verde según el asistente haya adquirido o no dichas opciones (con tooltips que permitan ver qué es cada cosa).

El controlador controllers/AttendeeController.php, en la acción actionIndex(), obtiene listados de los invitados y productos extra del evento actual, y se los pasa junto con el evento al modelo de búsqueda models/AttendeeSearch.php , mediante el método setEvent()

Dicho método (en realidad implementado en el modelo básico models/Attendee.php, del que AttendeeSearch hereda) construye listas internas de invitados, fotos y firmas asignadas a éstos, y productos extra a partir de la información recibida. A partir de esas listas se implementan los métodos getGuestFields() y getExtraProductsFields(), que devuelven, de todos los campos correspondientes a opciones adquiridas de fotos, firmas y productos extra, cuáles son los relevantes para los invitados y productos del evento actual.

La vista de listado views/attendee/index.php recupera dichas listas de campos, para crear los tíulos de los campos.

La visualización de las rejillas se delega en el componente components/AttendeeColumns.php , métodos mealsCol(), photosCol() y extraProductsCol()

#### Números de pedidos enlazados a las fichas de pedido

La aplicación permite registrar el número o números de pedido asociados a un asistente en la tienda online; y a la hora de mostrar esta información, cada número de pedido está enlazado a la ficha del mismo en dicha tienda.

Esto se implementa en el método orderNumbersCol de components/AttendeeColumns.php , invocado desde las vistas de listado views/attendee/index.php y de visualización de registro views/attendee/view.php

Models/AttendeeQuery.php tiene definido un método andFilterOrders(), que se utiliza desde AttendeeSearch(), para poder buscar por número de pedido (por subcadena, dado que el campo puede incluir varios números de pedido)

#### Verificación de errores

Al acceder al listado se comprueba que no haya registros con datos erróneos, y si los hay, se muestran para poder corregirlos.

Los casos de error contemplados son:

* Si un asistente es un padre / madre / guardián (es decir, está asignado como padre de otro asistente), debería tener asignado el número de teléfono en los datos del socio enlazado con el asistente.
* No debería haber dos asistentes con el mismo nombre de acreditación.
* Se verifica que los datos de alojamiento estén completos: si un asistente se aloja en habitación doble, tiene que tener un compañero de habitación, etc.
* Se verifica que los datos de alojamiento sean consistentes: un asistente no puede tener más compañeros de habitación que los que corresponden a la habitación en que se aloja.

El controlador controllers/AttendeeController.php obtiene los errores del modelo y los envía a la vista, en la acción actionIndex().

El modelo models/Attendee.php implementa el método checkErrors() que comprueba los datos y en su caso devuelve una lista de errores.

La vista views/attendee/index.php muestra los errores recibidos.

#### Campos con objetos relacionados enlazados a su ficha

Algunos de los campos de un asistente son asociaciones con otros objetos: idMember es un enlace a un registro de socios, idAttendeeRoommate1 / 2 / 3 y idAttendeeParent son enlaces a registros de asistentes. Cuando se visualizan los datos, dichos elementos aparecen enlazados a su ficha en la aplicación.

La vista views/attendees/view.php implementa estos enlaces.

#### Campos de fotos. firmas y productos extra dependen del evento

De entre todos los campos definidos para opciones de fotos, firmas y productos extra, la alta, visualización y modificación de un asistente sólo debe mostrar los que sean relevantes para el evento actual, y con los nombres correctos: es decir, que no ponga “foto del invitado 1”, sino el nombre del invitado.

El controlador controllers/AttendeeController.php, en las acciones actionView(), actionCreate() y actionUpdate(), obtiene el evento actual y las listas de invitados y productos extra, y se lo asignan al modelo mediante el método ya descrito setEvent().

Las vistas views/attendee/view.php y views/attendee/\_form.php obtienen los listados de campos de los métodos correspondientes del evento: getGuestFields() y getExtraProductsFields().

#### Actualizar fechas de última modificación sólo cuando se indica

Para mostrar correctamente los distintos informes, necesitamos varias fechas de última modificación para distintas áreas; y además, se puede dar el caso de que los datos cambien pero no queramos que se actualice la fecha, o viceversa. Por tanto hemos optado por un sistema semi-automático, en que el comportamiento por defecto es que las fechas sólo se actualizan si se modifican los campos, pero el usuario lo puede cambiar.

El controlador controllers/AttendeeController.php, en la acción actionUpdate(), comprueba si se han marcado las casillas de verificación correspondientes a cada fecha de modificación, y en cada caso, llama al método correspondiente del modelo para actualizarlas.

El modelo models/Attendee.php implementa una serie de métodos para guardar la fecha y hora actual en cada uno de los campos de última modificación:

* setUpdatedFlag()
* setUpdatedHotelFlag()
* setUpdatedBadgesFlag()
* setUpdatedBadgesTicketsFlag()

Esos métodos lo que hacen es marcar un flag interno en el objeto; los campos se modifican en base a estos flags en el método beforeSave() (que el framework llama automáticamente antes de guardar el registro).

Además, el modelo utiliza el componente TimestampBehavior, pero sólo asociado al evento de inserción.

En la vista de alta / modificación de registro views/attendees/\_form.php, los campos de última modificación se muestran como sólo lectura, y además con una casilla de verificación asociada a cada uno.

El módulo JavaScript checkboxes.js hace que dichas casillas se marquen automáticamente cuando se modifican los campos relevantes para cada una.

#### Campos de relación con autocompletado

Algunos de los campos de un asistente son asociaciones con otros objetos: idMember es un enlace a un registro de socios, idAttendeeRoommate1 / 2 / 3 y idAttendeeParent son enlaces a registros de asistentes. A la hora de dar de alta o modificar un asistente, se ha decidido que lo más cómodo es usar campos de autocompletado: se teclea parte del nombre y aparece una lista de registros coincidentes, se pulsa en el elegido y esto hace que se establezca el campo con el valor correspondiente (tanto el nombre visible como el identificador interno que se guardará como valor del campo en base de datos).

En el fichero de configuración config/web.php se define la ruta '/attendee/ajaxsearch/<term>' para poder pasarle el parámetro a la acción del controlador.

Los controladores controllers/MemberController.php y controllers/AttendeeController.php implementan la acción actionAjaxsearch($term) que recibe un término de búsqueda y devuelve una estructura JSON con los datos coincidentes.

En el caso de AttendeeController.php, delega dicha búsqueda en el método termSearch() del modelo models/Attendee.php Dicho método, a su vez, utiliza los métodos especializados del modelo de base de datos models/AttendeeQuery.php andFilterSearchMember(), además de andFilterEvent().

La vista de formulario de alta y modificación views/attendee/\_form.php define los campos relevantes mediante el componente de interfaz de usuario (widget) estándar de Yii llamado AutoComplete, con una serie de parámetros que recoge el módulo JavaScript autocomplete.js Éste implementa la asociación particular con los métodos del controlador correspondiente. La vista también define campos ocultos para almacenar el identificador interno del socio o asistente seleccionado; el nombre de dicho campo se le pasa al componente y lo recoge el JavaScript.

#### Informe de etiquetas de acreditaciones

A partir de los datos de asistentes se crea un informe con los nombres de acreditación de todos, pensado para imprimir en papel adhesivo, recortar cada nombre y pegarlo en las acreditaciones. Existe la posibilidad de imprimir sólo las nuevas incorporaciones o cambios desde la anterior impresión.

En el fichero de configuración config/web.php se definen las rutas '/attendee/reportbadgelabels/<afterprint>' y '/attendee/reportbadgelabels/<afterprint>/<showinfotickets>' , para poder pasar uno o dos parámetros a la acción.

La clase controlador controllers/AttendeeController.php define la acción actionReportbadgelabels ($afterprint, $showinfotickets). En dicha acción obtiene los datos correspondientes y se los pasa a una vista creada especialmente para esto.

Para obtener los datos se usan métodos definidos en la clase modelo de base de datos models/AttendeeQuery.php:

* andFilterEvent() -> filtra por el evento actualmente seleccionado
* notCanceled() -> elimina asistentes que hayan cancelado
* orderBadgeLabelReport() -> establece el orden necesario para el informe de etiquetas
* afterDate() -> filtra por los asistentes modificados tras una fecha determinada, según los parámetros indicados

El orden es:

* Primero, las acreditaciones de acompañantes de actores invitados
* A continuación, las acreditaciones pertenecientes a staff o con marca de especiales
* Después las demás, separadas entre fin de semana, viernes, sábado o domingo
* Dentro de cada bloque, ordenadas por nombre de acreditación
* Al final de todo, las acreditaciones en blanco con logo, si las hay

La vista es views/attendee/reportbadgelabels.php Dicha vista muestra la información con el formato adecuado para imprimirla. Puede haber algunas etiquetas extra añadidas a mano (ACADI, droides R2).

La vista enlaza el módulo JavaScript reportcheckdimensions.js, que verifica que las etiquetas no se pasen de ancho y no quepan en las acreditaciones. Existe la posibilidad de marcar algunas para que tengan la letra más pequeña.

#### Informe de acreditaciones

Este informe muestra una lista de los acreditados y sus opciones, con algunas variaciones:

* Lista completa de acreditados con nombre real, nombre de acreditación, tipo de entrada y observaciones
* Lista de acreditados que tengan tickets de firmas, fotos o cartones
* Lista de acreditados que tengan cenas o comidas, con sus opciones especiales
* Lista de acreditados de grandes asociaciones que van en lista separada

En el fichero de configuración config/web.php se define la ruta '/attendee/reportbadges/<detailed>' para poder pasarle un parámetro a la acción del controlador. Este parámetro controla cuál de los informes se muestra.

La clase controladora controller/AttendeeController.php define la acción actionReportbadges ($detailed) que recupera los datos y se los pasa a la vista.

Para ello usa determinados filtros definidos en el modelo de base de datos models/AttendeeQuery.php:

* andFilterEvent()
* orderBadgeReport() -> establece los filtros y el orden necesario para los informes. Ambos son diferentes según el tipo de informe, por lo que recibe el parámetro $detailed

Hay vistas distintas definidas según el tipo de informe:

* views/attendee/reportbadges.php: informe básico de acreditados.
* views/attendee/reportbadgesdet.php: informe de acreditados con fotos, firmas o cartones. Para los colores de fondo utiliza los métodos del modelo models/Attendee.php:
  + getGuestFieldColor()
  + getExtraProductFieldColor()
* views/attendee/reportbadgesclubs.php: informe de acreditados en asociaciones con lista separada. Muestra una cabecera por cada asociación para distinguirlas fácilmente.
* Views/attendee/reportbadgesmealstable.php: informe de acreditados con cenas o comidas.

#### Informe para el hotel

Otro informe más está diseñado a medida con la información que necesita el hotel: el listado de reservas de habitaciones, con tipos, nombres, fechas y observaciones; y las cifras de cenas y comidas con las observaciones necesarias para configurar los menús y opciones especiales.

En el fichero de configuración config/web.php se define la ruta '/attendee/reporthotel/<aftersend>' para poder pasarle un parámetro a la acción del controlador.

La clase controladora controller/AttendeeController.php define la acción actionReporthotel ($aftersend) que recoje y coteja los datos necesarios y se los pasa a la vista. El parámetro $aftersend controla si se va a generar un informe que tenga marcadas las actualizaciones desde la versión enviada anteriormente.

Algunas de las funcionalidades de los modelos que utiliza esta acción son:

* Attendee::getAttendeeRooms(): obtiene toda la información de reservas de habitaciones, uniendo los datos de reservas de los compañeros para habitaciones de más de una persona.
* AttendeeQuery::andFilterLodging(): filtra para devolver solo asistentes con reserva de habitación.
* AttendeeQuery:: andCocktail(): filtra los asistentes que tengan cena cóctel.
* AttendeeQuery:: andGala(): filtra los asistentes que tengan cena de gala.
* AttendeeQuery::andSaturdayLunch(): filtra los asistentes que tengan comida del sábado. Lo mismo con andSundayLunch() y andSundayDinner().

La vista es views/attendee/reporthotel.php

#### Vista de Acreditaciones

Se ha creado una vista adecuada para utilizarse en el propio evento, en la mesa de acreditaciones. Tiene limitadas bastantes funcionalidades, para concentrarse en las necesidades concretas de ese perfil de usuario: buscar los datos de reserva de un asistente, teniendo en cuenta su nombre de acreditación.

Se ha definido un rol llamado ‘Desk’, que define los permisos mínimos para gestionar esto (básicamente, acceso de sólo lectura a Asistentes) y además identifica a qué usuarios se le debe aplicar la vista restringida.

En la clase controladora controllers/AttendeeController.php , el método actionIndex() verifica si el usuario tiene el rol Desk; y en ese caso, activa la vista indexdesk en vez de index. Igualmente, en actionView(), activa la vista viewdesk en vez de view para los usuarios con rol Desk.

La vista views/attendee/indexdesk.php se diferencia de views/attendee/index.php en:

* Tiene desactivadas las funcionalidades de añadir, modificar o borrar asistentes.
* Están desactivados los filtros por evento (se aplica un filtro fijo por el evento actual), estado, tipo de acreditación y procedencia
* Se oculta la columna Acreditación especial
* Se añade la columna Nombre real además del Nombre de acreditación
* Se eliminan los enlaces a los pedidos en la tienda

La vista views/attendee/viewdesk.php se diferencia de views/attendee/view.php en:

* Se eliminan los botones de modificar y borrar
* Se eliminan los enlaces a la ficha del evento y el socio
* Se eliminan los enlaces a los pedidos en la tienda
* Se elimina la información de alojamiento, fechas de última modificación, y observaciones internas y de comidas

El método rules() en el modelo de búsqueda models/AttendeeSearch() devuelve datos levemente diferentes según el perfil, lo que es necesario para que en la vista de acreditaciones no aparezcan filtros ni cajas de búsqueda para las columnas Evento y Tipo de entrada.

### Objetos de aplicación – medio de prensa

#### Procedencia

El campo idSource se puede asociar con una procedencia que se haya definido para acreditar miembros de este medio. De momento es sólo informativo, pero en el futuro se puede usar para funcionalidades que requieran tener asociados los socios a su medio.

Tanto en el filtro de búsqueda (vista views/press/index) como en el alta y modificación (vista views/press/\_form), el campo se muestra como un desplegable; las opciones son aquellas procedencias cuyo nombre comience por “Prensa”. Se le da contenido en la clase modelo app\models\Press.php, método getSources.

Además, se define la propiedad sourceName; para darle valor, todos los accesos a base de datos que recuperen datos de medios de prensa añaden una cláusula Join para unir la tabla de procedencias y recuperar el nombre: clase modelo app\models\PressSearch.php, método search(), y clase controladora app\controllers\PressController.php, métodos findModel() y actionExport()

#### Exportación

Desde la vista views/press/index se enlaza la acción actionExport de la clase controladora app\controllers\PressController.php Devuelve un pseudo-Excel con los datos de todos los medios. Por defecto filtra sólo los que tengan marcado el consentimiento; hay una segunda opción para filtrar por los que no lo tienen marcado (misma acción con otro valor de parámetro), para el mailing masivo descrito en la siguiente sección.

#### Gestionar consentimiento

Debido a la nueva ley RGPD, se necesita recabar el consentimiento expreso de los medios de prensa para poder enviarles correos electrónicos. En el futuro lo solicitaremos expresamente, pero para los existentes, haremos un emailing masivo en el que solicitaremos consentimiento para las siguientes comunicaciones, proporcionando un enlace que al pincharlo recogerá el consentimiento del medio.

#### Recoger consentimientos

La idea es que, a todos los medios que no hayan dado consentimiento, se les envía un e-mail solicitándolo. Dicho e-mail contendrá un enlace al módulo que se va a describir, que incluye el correo electrónico y una clave de verificación; siempre que se localice un medio para el que coincidan esos dos datos, se actualizará el consentimiento y mostrará un mensaje de verificación.

El campo keyCheck es de sólo lectura en el alta o modificación de medio (vista views/press/\_form). Cuando se da de alta un medio, se le asigna un keyCheck único y aleatorio (método setKey de la clase modelo app\models\Press ).

La acción actionConsent de la clase controladora app\controllers\PressController es a la que enlazará el correo que se envíe a los medios. No requiere autenticación (al estar indicada en el método getFreeAccessActions() de la clase controladora) y tiene un layout distinto para parecer parte de la web de CifiMad. Recibe el correo y la clave de verificación; si encuentra un medio para el que coincidan los dos valores, actualiza su consentimiento. La vista definida en views/press/consent muestra un mensaje de confirmación si el consentimiento se ha actualizado, o un mensaje de error si el socio no se encuentra.