IES Gran capitán
Módulo de proyecto integrado
2015-2016

Gestor de biblioteca escolar

Alejandro Gutiérrez Lozano
2º C.F.G.S. Desarrollo de aplicaciones web

Índice

1. Introducción

En este proyecto se va a realizar un gestor para una biblioteca particular de un centro educativo.
El comportamiento de una biblioteca se compone generalmente de unos libros que se prestan a unos usuarios durante un tiempo predeterminado. Los usuarios serán el alumnado y profesorado del centro.

Referente a los libros, tenemos información sobre su título, autor, ISBN etc...
La biblioteca está distribuida en áreas cuyo fin es localizar los ejemplares de los libros que tienen una temática similar de manera más sencilla.
De cada libro se suele tener varios ejemplares para prestar, dicho ejemplares son los prestados a los usuarios. En todo momento es importante saber el estado de conservación de los ejemplares, el cual depende de circunstancias varias, como el uso que se le han dado o la antigüedad.

Los usuarios solicitan un ejemplar durante un tiempo predeterminado.
En dicho préstamo es obligatorio saber la fecha inicial que indica el día que se hizo el préstamo y la fecha final indicando que día el usuario tiene que devolver del ejemplar.
Los usuarios tienen un máximo de préstamos, para evitar un abuso del sistema.

En los centros educativos también hay unos préstamos particulares, que tienen como fin evitar que los padres compren libros escolares y el gasto que ello supone. Para conseguir ese fin, el centro posee ejemplares de dichos libros y los presta a los alumnos durante el curso lectivo, una vez terminado el curso, los alumnos devuelven los libros para que puedan ser reutilizados.

Los responsables de la biblioteca son los administradores, que controlan el buen control y funcionamiento del sistema, podrá haber varios responsables y ellos llevan el control total de los datos de la biblioteca, y realizan operaciones con ellos.

Suele haber un límite de administradores, pero no hay un número por defecto, son ellos mismos los que regulan ese límite.

Además, suele haber un registro externo donde se introduce de manera automática cualquier cambio que hubiera en la biblioteca, tanto como modificación de ejemplares, como inserción de un usuario nuevo, se hace con el fin de tener algún lugar para apuntar los cambios realizados.

2. Objetivos

La aplicación del gestor tiene los siguientes objetivos principales:

-**Gestión de libros**. Incluye la creación de los libros así como editarlos u eliminarlos. Hay que tener en cuenta, entre otros aspectos, que al eliminar un libro se eliminan también todos los ejemplares que proceden del libro eliminado.

-**Gestión de préstamos**. Como poder observar los prestamos activos u inactivos, editar alguna fecha o la introducción correcta de estos de manera sencilla.
Dentro de este objetivo, introducimos los préstamos gratuitos, que consisten en préstamos que tienen un determinado libro de texto para un usuario. Ese libro se devuelve una vez finalizado el curso actual, para prestarlo a un nuevo usuario.

-**Gestión de los alumnos**, los alumnos serán tratados como usuarios del sistema. Se guardará información sobre ellos y se podrá dar de alta, modificar sus datos o dar de baja si ellos lo desean. También habrá que contar con varios administradores que realicen las gestiones de los préstamos, de los usuarios y los libros.

-**Uso de framework de desarrollo**, tiene varias finalidades, siendo una de ellas el uso del modelo MVC (Modelo-vista-controlador) y poder realizar de manera más sencilla y rápida el lenguaje de programación que estemos utilizando ampliando, la mayoría de las veces sus capacidades.

-**Usabilidad**. Siempre tenemos en mente nuestro objetivo de la usabilidad de la aplicación, facilitando lo máximo posible al administrador en este caso, como configurar y realizar las gestiones pertinentes dentro de la aplicación web.

Algunos objetivos fueron surgidos durante la creación del gestor de biblioteca los cuales son:

-**Sistema de búsqueda.** Que pueda realizar de manera sencilla una consulta específica a la base de datos y devolver el resultado en forma de tabla.

-Sistema de incidencias. Relacionadas con un préstamo. El objetivo es añadir o ver cualesquiera incidencias que hubiera en algún préstamo de un ejemplar a un usuario, como la modificación de la fecha de finalización o la devolución de un préstamo en un mal estado.

3. Estudio previo

3.1 Estado actual

El estado actual de la aplicación existente es la siguiente:
Actualmente existe un programa que realiza las funciones de gestor de biblioteca de manera eficiente. El programa, además de alta de libros, préstamos, etc… Tiene funciones adicionales como lectura de libros por código que añade automáticamente la información del libro en el programa, da la oportunidad de añadir una gran cantidad de datos de los libros u ejemplares, etc…
Debido a que el propietario es la Junta de Andalucía, el software es propietario y se necesita informar de la biblioteca del centro educativo para que puedas bajar y usar el programa.

También hay un gestor de biblioteca escrito en PHP puro, pero al ser una aplicación grande, cuando tienes que ir modificando o actualizando el código se vuelve muy confuso. La ventaja respecto al programa actual existente es que permite su uso libremente y modificarlo.

3.2 Estudio de soluciones existentes

Hay dos soluciones que pueden servir para nuestro propósito de mejorar la aplicación web de la biblioteca.

La primera es actualizar la aplicación existente de PHP de manera que este basado en el diseño del modelo vista-controlador, para ello, habría que, con el código existente organizarlo, extraer la parte visible de la aplicación, la parte del controlador que envía y obtiene información y la parte del modelo de datos que trabaja y realiza todas las consultas a la base de datos. El código, al ser de unas dimensiones muy grandes, ver que parte del código realiza cada función estando todo mezclado hace que organizarlo se convierta en un proceso muy largo y complicado.

La segunda, es usar un framework que nos facilite la separación física y lógica de los datos desde el principio. Esto supone una desventaja al tener que empezar la aplicación ya existente desde el principio, pero, tiene dos ventajas, evitar el proceso de separar el código escrito que nos puede hacer perder incluso más tiempo de hacer la aplicación desde el principio y facilitar la creación de nuevas vistas o consultas a la biblioteca a la hora de actualizar la aplicación además de una mejor organización del código.

4. Plan de trabajo

Mi plan de trabajo será el siguiente:

Búsqueda de framework e instalación. En esta parte buscaré una lista de posibles frameworks con los que se pueda realizar la aplicación deseada y, la instalación de dicho framework. La duración aproximada de la instalación del framework, su configuración y observar que funciona correctamente estimo que será de 2 días.

Manejo e introducción del framework. Como su nombre indica, en esta parte empezaré a trabajar con el framework y realizar una pequeña aplicación, para tener manejo suficiente para comenzar a realizar la aplicación del gestor de biblioteca. La duración estimada para obtener los conocimientos necesarios para trabajar con el framework de manera eficiente será alrededor de 1 semana.

Búsqueda de información y modelo lógico. El objetivo de este punto es obtener información sobre los datos que se deben almacenar en una biblioteca, una vez obtenidos esos datos, habrá que realizar un modelo entidad-relación que posteriormente pasará a la base de datos y se almacenará la información de la aplicación del gestor de biblioteca. El tiempo estimado en realizar el modelo entidad-relación, buscar los datos y solventar problemas que puedan surgir es de unas 2 semanas.

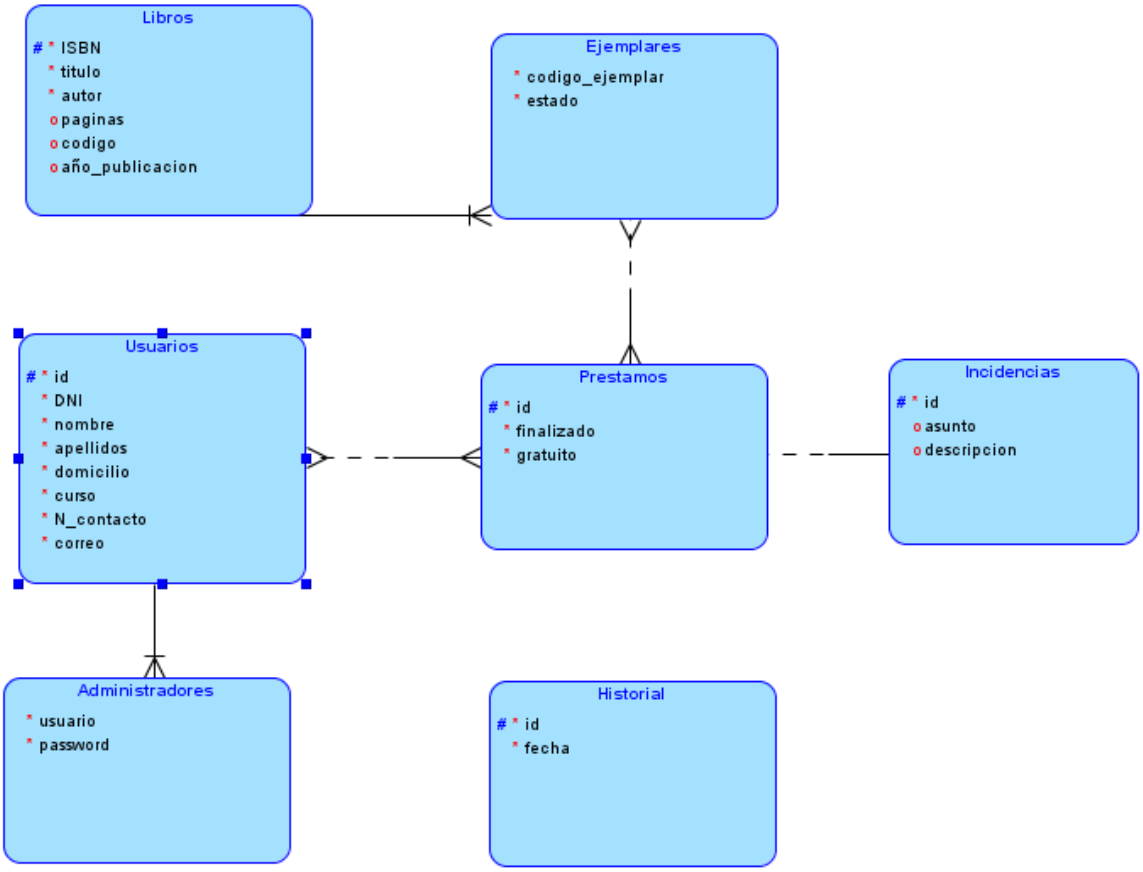
Diseño y estructura de la aplicación. Es la parte central de plan de trabajo, dónde haremos los distintos apartados y daremos forma al proyecto. Aquí entra toda la parte de programación del framework, de la página y la base de datos. Una vez finalizado esta parte deberemos tener una aplicación estable que contenga, al menos, la gestión de los usuarios, los préstamos, los libros, los ejemplares. Forma la parte principal del proyecto y con una duración aproximada de 6 semanas.

Corregir errores y comprobar que funcione adecuadamente. Por último tenemos la fase donde corregir o añadir nuevas funcionalidades a la aplicación web que están fuera de los objetivos y ejecutar la aplicación ya en el entorno de trabajo. La duración de esta fase de desarrollo abarca unas 2-3 semanas, dependiendo del tiempo usado en las anteriores partes.

5. Diseño

5.1 Diseño general

La aplicación del gestor tiene como diseño lógico el esquema relacional siguiente. El esquema fue sufriendo transformaciones hasta terminar en su forma actual:



Algunas ideas desechadas conforme a la evolución del diseño son las siguientes:

- **Un sistema restrictivo de faltas** donde según su número y la gravedad de estas bloquearan automáticamente al usuario. No era una buena idea un sistema arbitrario de eliminación automática sin tener en cuenta factores externos. La mejor opción es dejar que los administradores prohíban a los usuarios realizar préstamos cuando crean oportuno.

- **Dividir los usuarios en administradores y alumnos**. Duplicaba todos los campos de los usuarios salvo dos y el sistema era enrevesado. El modelo actual es más sencillo y más sencillo que el anterior.
- **Los usuarios podían entrar y ver sus préstamos**, pero, debido al funcionamiento de una biblioteca, dónde los administradores son los únicos que deben acceder al sistema, no tenía sentido.
- **Las incidencias tenían un estado**, dependiendo de la gravedad de la falta, al contar con una descripción es más sencillo y menos arbitrario incluir optativamente el estado que añadir una enumeración, que, no contará con todos los tipos que puede tener la incidencia.

.El diseño conceptual de la aplicación tiene los siguientes objetivos:

- La creación, modificación o eliminación de cualquier libro o ejemplar que se desee, de manera fácil y sencilla.

- Gestión de los préstamos, pudiendo observar los plazos de devolución de los libros de manera simple, permitir modificar el plazo de entrega u observar qué cliente tiene tal libro.

- Gestión de los usuarios, cuya creación se encargaran los administradores, se podrán modificar o eliminar si ellos lo ven necesario.

- Deberán hacer falta uno o varios administradores que realicen las gestiones de los préstamos, como observar los préstamos o añadirlos.

- Creación de un historial cuya función es registrar cualquier acción que se realice dentro de la aplicación y guardarla.

- Sistema de incidencias. Para anotar cualquier dato o incidente de importancia de algún préstamo. Los administradores serán los que, en función del número de faltas, o la gravedad de las incidencias no permitirá a dicho usuario realizar préstamos.

5.2 Diseño detallado

El diseño de la base de datos cuenta con las siguientes tablas en dónde se guardarán los datos introducidos por la aplicación.

- **Libros**, tendrán una cuya clave será el **ISBN**, será necesario al menos introducir los siguientes datos para crear un libro: el ISBN, un título y el autor.

Adicionalmente, se podrá añadir el número de páginas, su CDU (Clasificación Decimal Universal) y el año de publicación.

- **Ejemplares**, tendrán como clave principal el ISBN del libro el cual les corresponde y un código que lo identifica, además se podrá incluir un estado de conservación de manera opcional.

- **Usuarios**, Se necesitan los siguientes datos para crear un usuario: DNI, nombre, apellidos, domicilio, curso, un número de contacto y un correo electrónico.
  La clave primaria de la tabla es el **id**, el cual será un número creado aleatoriamente y el DNI del usuario será único.

- **Administradores** tienen un usuario y una contraseña, con la entrarán al sistema y tienen la capacidad de realizar todas las gestiones del sistema

- **Préstamos**. Se almacenará un préstamo por cada ejemplar que se desee. Los usuarios podrán hacer varios préstamos a la vez, pero tendrán un límite. Los préstamos se relacionarán con los usuarios y los ejemplares. Para identificar al préstamo, tendrán asociado un id que se autoincrementa cada vez que se incluya un préstamo, una fecha de inicio que se insertará automáticamente al momento de añadir el préstamo y una fecha de finalización, esta fecha indica el día de devolución del libro, podrá tener una fecha en concreto, en caso de un préstamo personal, o, se añadirá automáticamente en caso de los préstamos gratuitos. Estos últimos siempre tendrán la duración de un año de curso escolar.
- **Las incidencias** están asociadas a un préstamo. Tendrán un id único, un nombre y descripción. Los administradores pueden añadir incidencias o eliminarlas aunque no editarlas.

- **El historial** es una tabla no relacionada con ninguna dónde se introducirá de manera automática cualquier cambio que se realice dentro del sistema. Como atributos, tiene un id único que va incrementándose automáticamente, una fecha de la hora y día que se produjo el acceso a la base de datos, y una pequeña descripción sobre el cambio que se realizó.

La aplicación web está basada en el diseño de arquitectura web el modelo vista-controlador. El objetivo de esta arquitectura es separar la lógica de la aplicación de la interfaz del usuario.

Al estar basada en dicha arquitectura, tiene tres partes bien diferenciadas.
Los datos, los cuáles serán manipulados en su totalidad por un archivo en cuyo código se realizarán todas las operaciones que tienen como fin acceder y manipular la información registrada en la base de datos.
El controlador, será la parte principal de la aplicación, esta parte se encarga de recoger los datos que envía el usuario y mostrar la interfaz dependiendo de dichos datos o el enlace que se indica.
La interfaz, es la parte mostrada al usuario final, en este caso, los administradores. Dependiendo de los datos que haya en la base de datos y las acciones llevadas a cabo por los administradores se muestra la información deseada.

6. Implementación

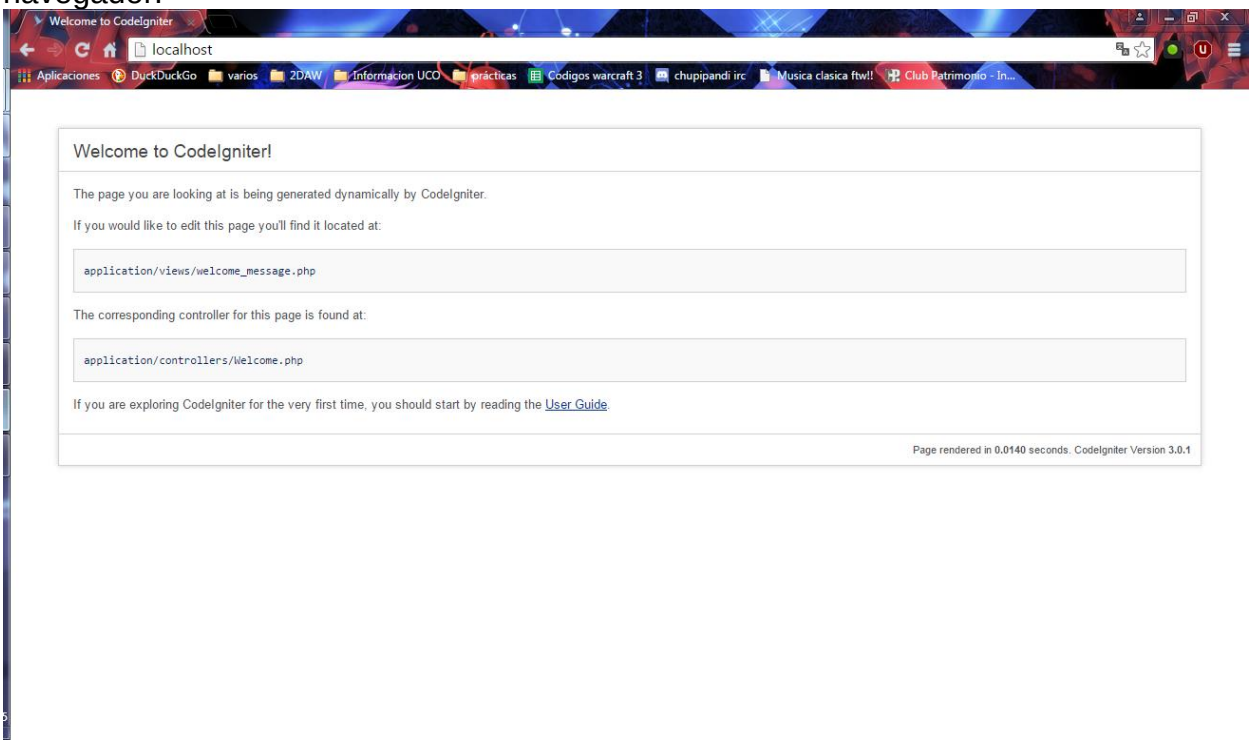
Instalación del framework utilizado

El framework, que utilizaré será CodeIgniter. La página oficial del framework de desarrollo es la siguiente: https://codeigniter.com/ y como servidor web, usare el Apache Server, además usaré boostrap para el estilo y diseño para móviles.
Para instalar el framework, se descarga de la página oficial y se extrae el contenido del archivo.zip en la localización que se desee y, una vez hecho, con el servidor web que hayamos montado, modificamos la directiva “DocumentRoot” adónde hallamos descomprimido el archivo.

```
#
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
#
DocumentRoot "D:/2DAW/Proyecto/biblioteca"
<Directory "D:/2DAW/Proyecto/biblioteca">
```

Archivo de configuración del servidor apache Httpd.conf

Si la instalación la hemos hecho correctamente, nos saldrá lo siguiente en el navegador.



Pagina por defecto del framework codeIgniter

Indicando que es una página por defecto y donde está localizado el controlador de la página.

Configuración básica del framework

La configuración básica es lo siguiente:
En la carpeta application/config, buscar el archivo config.php y modificar la base_url.
Esta url será la principal de nuestra aplicación.

```
/*
|--------------------------------------------------------------------------
| Base Site URL
|--------------------------------------------------------------------------
|
| URL to your CodeIgniter root. Typically this will be your base URL,
| WITH a trailing slash:
|
|   http://example.com/
|
| If this is not set then CodeIgniter will try guess the protocol, domain
| and path to your installation. However, you should always configure this
| explicitly and never rely on auto-guessing, especially in production
| environments.
|
*/
$config['base_url'] = 'http://www.bibliotecaGranCapitan.org/';
```

Archivo de configuración de la aplicación config.php.

la conexión a la base de datos se realiza desde el archivo database.php,
dicho archivo está localizado dentro de nuestro proyecto en “\application\config”.

```
$active_group = 'default';
$query_builder = TRUE;

$db['default'] = array(
    'dsn'   => '',
    'hostname' => 'localhost',
    'database' => 'biblioteca',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => TRUE,
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
    'save_queries' => TRUE
    'username' => [redacted],
    'password' => [redacted],
);
```

Archivo de configuración de la base de datos database.php.

Si lo hemos hecho adecuadamente, al intentar acceder posteriormente a la base de datos no debe dar ningún error.

Mostrar una página conectándose a una base de datos

Para mostrar una página hace falta añadir al controlador de la aplicación la vista que queramos mostrar y, crear la página.

Comencemos creando el archivo biblioteca.php dentro de controller y añadiendo lo siguiente:

```php
class biblioteca extends CI_controller{
    public function __construct(){
        parent::__construct();
        $this->load->library('form_validation');
        $this->load->model('biblioteca_model');
        $this->load->helper('url_helper');
        $this->load->helper('url');
        $this->load->helper('form');
        $this->load->helper('html');
    }

    public function login(){
        $data['title'] = 'Login';
        $this->load->view('templates/header',$data);
        $this->load->view('biblioteca/login',$data);
        $this->load->view('templates/footer',$data);
    }
```

Archivo biblioteca.php dentro de application/controllers

El archivo se compone de una clase que extiende de otra llamada CI_controller que tiene todos los métodos necesarios para hacer de controlador de la aplicación, la clase del archivo hereda el constructor de CI_controller, por lo cual, este archivo tendrá todos los métodos de su padre.

El siguiente paso es crear el archivo que queremos observar, en nuestro caso hay tres. El primero es header.php localizado en “application/views/templates”, para evitar crear el html y el head en todas las páginas, se puede extraer en un archivo aparte e, insertarlo antes de la página principal.
El código del archivo es:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="Content-Type" context="text/html; charset=utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">

    <title>Biblioteca</title>

    <link href="<?php echo base_url('assets/css/bootstrap.min.css') ?>" rel="stylesheet">
    <link href="<?php echo base_url('assets/css/font-awesome.min.css') ?>" rel="stylesheet">
</head>
<body>
    <div class="container-fluid">
    <h1>
        <?php echo $title; ?>
    </h1>
```

Archivo header.php

Tiene las etiquetas de meta, los link de bootstrap y, una variable $title que posteriormente explicaré cual es su finalidad.

El archivo “login.php”, que se encuentra en “application/views/biblioteca”.
El contenido del archivo es el siguiente:

```
<?php echo validation_errors();

$atributtes = array(
    'class' => 'form-inline',
    );
echo form_open('biblioteca/login',$atributtes);
        $usuario = array(
            'name' => 'usuario',
            'value' => 'admin1',
            'class' => 'form-contol',
            );
        $password = array(
            'name' => 'password',
            'value' => 'admin',
            'class' => 'form-contol',
            );
        $envio = array(
            'name' => 'entrar',
            'value' => 'entrar',
            'class' => 'form-contol',
            );
     echo '<label>user</label>'.form_input($usuario)
     .'<label>password</label>'.form_password($password)
     .form_submit($envio);
     ?>
```

Archivo login.php

El archivo contiene varias funciones que facilitan la la tarea como se puede observar. Se puede hacer un array con los atributos que tendrá los inputs y el framework automáticamente creara esos inputs con los atributos pasados.

La página simplemente tiene un formulario (form_open) dos inputs, uno normal y otro de contraseña y un botón submit que envía los datos al servidor.

El footer contiene lo siguiente:

```
        <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
    <!-- Include all compiled plugins (below), or include individual files as needed -->
    <script src="<?php echo base_url('assets/js/bootstrap.min.js') ?>"></script>
    <br/><em>Alejandro Gutiérrez Lozano&copy; 2015</em>
    </body>
</html>
```

Archivo footer.html

Contiene el javascript que necesita bootstrap para funcionar adecuadamente y cierra las etiquetas del body y html.

El siguiente paso es añadir la vista al controlador, se ha hecho en el 1º paso, justamente son estas líneas:

```
$this->load->view('templates/header',$data);
$this->load->view('biblioteca/login',$data);
$this->load->view('templates/footer',$data);
```

Esas tres líneas cargan los tres archivos que hemos creado y lo muestra por pantalla.
Además de esta línea

```
$data['title'] = 'Login';
```

Si recordaís el $title del header. CodeIgniter permite pasar parámetros del controlador a las vistas con un array asociativo reservado llamado $data.
En nuestro caso, además de pasar los datos recibidos por la biblioteca, lo haremos para mostrar un titulo diferente en cada vista que realicemos.

Por último, debemos añadir a la ruta la dirección en a cual queremos que se muestre la página en este caso es login. Los archivos de ruta se localizan en el archivo routes.php dentro de “application/config”, la carpeta donde esta config.php y database.php modificados anteriormente.
El archivo contiene lo siguiente:

```
| -------------------------------------------------------------------
| RESERVED ROUTES
| -------------------------------------------------------------------
|
| There are three reserved routes:
|
|   $route['default_controller'] = 'welcome';
|
| This route indicates which controller class should be loaded if the
| URI contains no data. In the above example, the "welcome" class
| would be loaded.
|
|   $route['404_override'] = 'errors/page_missing';
|
| This route will tell the Router which controller/method to use if those
| provided in the URL cannot be matched to a valid route.
|
|   $route['translate_uri_dashes'] = FALSE;
|
| This is not exactly a route, but allows you to automatically route
| controller and method names that contain dashes. '-' isn't a valid
| class or method name character, so it requires translation.
| When you set this option to TRUE, it will replace ALL dashes in the
| controller and method URI segments.
|
| Examples: my-controller/index -> my_controller/index
|       my-controller/my-method -> my_controller/my_method
*/
$route['default_controller'] = 'welcome';
$route['404_override'] = '';
$route['translate_uri_dashes'] = FALSE;
```

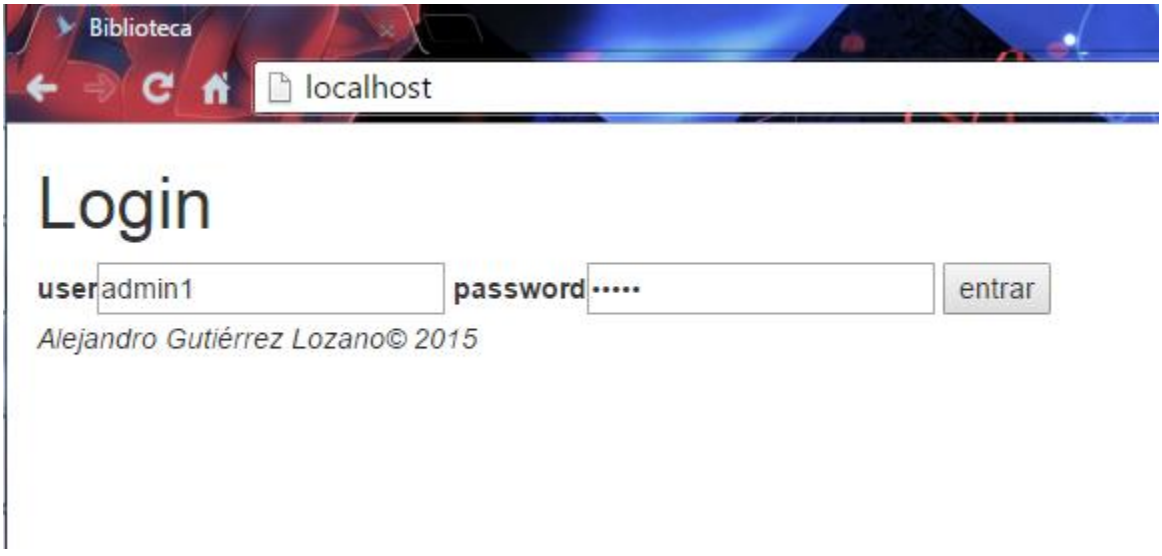
Archivo routes.php por defecto

Contiene información útil sobre el archivo y como se configura.
Y lo modificamos para que quede como se muestra a continuación:

```
| -------------------------------------------------------------------
| RESERVED ROUTES
| -------------------------------------------------------------------
|
| There are three reserved routes:
|
|   $route['default_controller'] = 'welcome';
|
| This route indicates which controller class should be loaded if the
| URI contains no data. In the above example, the "welcome" class
| would be loaded.
|
|   $route['404_override'] = 'errors/page_missing';
|
| This route will tell the Router which controller/method to use if those
| provided in the URL cannot be matched to a valid route.
|
|   $route['translate_uri_dashes'] = FALSE;
|
| This is not exactly a route, but allows you to automatically route
| controller and method names that contain dashes. '-' isn't a valid
| class or method name character, so it requires translation.
| When you set this option to TRUE, it will replace ALL dashes in the
| controller and method URI segments.
|
| Examples: my-controller/index -> my_controller/index
|       my-controller/my-method -> my_controller/my_method
*/
$route['default_controller'] = 'biblioteca/login';
$route['biblioteca/login'] = 'biblioteca/login';
```

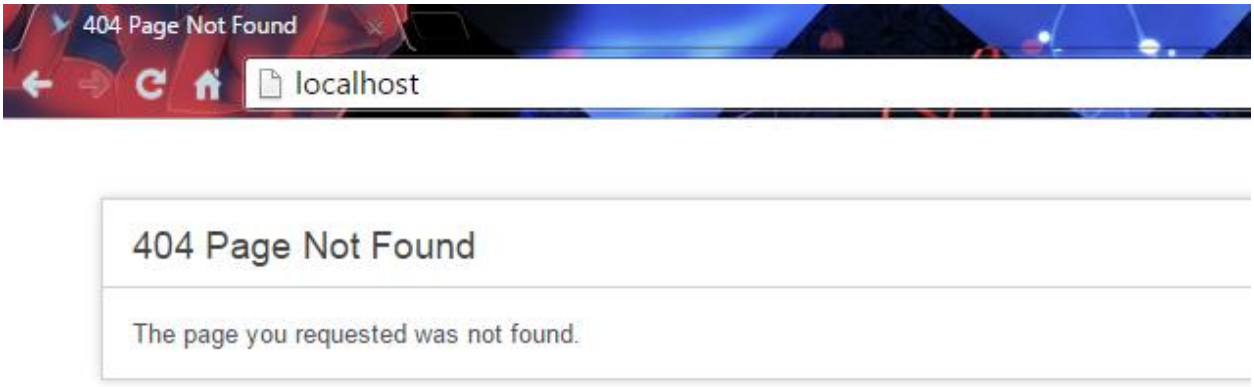
Archivo routes.php modificado

Si lo hemos hecho bien, al reiniciar el servicio apache para que actualize los cambios, se debe ver esta página en el navegador al poner localhost o localhost/biblioteca/login:



Página de inicio de la aplicación web

En caso de no localizar la página, se mostrara una página como esta:



Prosigamos, lo siguiente es conectar a la base de datos para ver si al introducir los datos en el formulario encuentra algún resultado.
Para ello, CodeIgniter nos ofrece un par de funciones para facilitarnos el trabajo.
Modifica el controlador, biblioteca.php y añade lo siguiente a lo que había:

```php
public function login(){
    $data['title'] = 'Login';

    if(!$this->input->post('entrar')){

        $this->load->view('templates/header',$data);
        $this->load->view('biblioteca/login',$data);
        $this->load->view('templates/footer',$data);
    }

    if($this->input->post('entrar')){
        $this->form_validation->set_rules('usuario','usuario','required');
        $this->form_validation->set_rules('password','password','required');
        $usuario = $this->input->post('usuario');
        $password = $this->input->post('password');
        $comprobacion = $this->biblioteca_model->comprobarUser($usuario,$password);

        if($this->form_validation->run()===FALSE || empty($comprobacion)){
            $this->load->view('templates/header',$data);
            $this->load->view('biblioteca/login',$data);
            $this->load->view('templates/footer',$data);
        }
    }
}
```

Archivo biblioteca.php modificado

Comienza con una condición (¡$this->input->post(‘entrar’)). Que comprueba si el botón cuyo nombre es ‘entrar’ ha sido pulsado. Al entrar en la página no estará pulsado así qué inicialmente se mostrará la página vista anteriormente.
La siguiente condición indica que el botón ha sido pulsado. Añadimos un par de reglas, que indica que ambos campos son requeridos y obtenemos ambos inputs.

Por último tenemos un método que comprobará si en la base de datos existe un usuario/contraseña que hayamos introducido.
El archivo biblioteca_model esta dentro de la carpeta “application/models” y contiene lo siguiente:

```php
class Biblioteca_model extends CI_Model{

    public function __construct(){
        //carga la base de datos
        $this->load->database();
        $this->load->dbforge();
    }

    public function comprobarUser($user,$pass){
        $this->db->where('usuario',$user);
        $this->db->where('password',$pass);
        $query = $this->db->get('administradores');
        return $query->result_array();
    }
```

Archivo biblioteca_model

Carga la configuración de la base de datos que, en caso de que sea incorrecta nos saldrá un error al intentar acceder a ella y una función que facilita la realización de consultas a la base de datos.

La función, busca en la tabla administradores que haya una igualdad de usuario/contraseña.

Visto anteriormente, si no se cumple o, si la variable esta vacía que significa que no hay ningún resultado, el sistema no permitirá el acceso y volverá al estado inicial.
En otro caso, irá a la página inicial del sistema, en mi caso es dónde se encuentra la lista de los libros.

La función de la vista login al completo es el siguiente:

```php
public function login(){
    $data['title'] = 'Login';

    if(!$this->input->post('entrar')){

        $this->load->view('templates/header',$data);
        $this->load->view('biblioteca/login',$data);
        $this->load->view('templates/footer',$data);
    }

    if($this->input->post('entrar')){
        $this->form_validation->set_rules('usuario','usuario','required');
        $this->form_validation->set_rules('password','password','required');
        $usuario = $this->input->post('usuario');
        $password = $this->input->post('password');
        $comprobacion = $this->biblioteca_model->comprobarUser($usuario,$password);

        if($this->form_validation->run()===FALSE || empty($comprobacion)){
            $this->load->view('templates/header',$data);
            $this->load->view('biblioteca/login',$data);
            $this->load->view('templates/footer',$data);
        }
        else{
            $data['title'] = 'Lista de libros';
            $data['libros'] = $this->biblioteca_model->get_biblioteca();
            $this->load->view('templates/headerAdmin',$data);
            $this->load->view('biblioteca/index',$data);
            $this->load->view('templates/footer',$data);
        }
    }
}
```

Función login dentro del archivo biblioteca.php

En caso de que encuentre al menos un resultado y se introduzca datos, mostrará otro header que será un menú en la parte superior y el index dónde se encuentra los datos de los libros.

Mostrar datos de una consulta realizada en la base de datos

El siguiente paso es hacer el index principal, en mi caso, es la lista de libros.
Hace falta realizar 3 pasos para que muestre la lista de libros.
Añadir el index en el controlador, para ello, nos vamos al archivo biblioteca.php y creamos una nueva función llamada index y escribimos el siguiente código:

```php
//Funcion de la pestaña de la lista de libros
public function index(){
    $data['title'] = 'Lista de libros';
    $data['libros'] = $this->biblioteca_model->get_biblioteca();
        $this->load->view('templates/headerAdmin',$data);
        $this->load->view('biblioteca/index',$data);
        $this->load->view('templates/footer',$data);
}
```

Función index de biblioteca.php

Le pasamos dos parámetros a la vista.
title, que será el título que tenga la página.
libros cuyo contenido será una lista de todos los libros insertados en la base de datos.

En biblioteca_model añadimos la función get_biblioteca(). Dicha función tiene como objetivo obtener todos los datos de todos los libros de la base de datos. La cosulta en MySQL es igual a “select * from libros”.
CodeIgniter nos ofrece la posibilidad de usar funciones de manera que las consultas sean más fáciles de realizar o podemos seguir usando código MySQL.
En nuestro caso, obtenemos los datos con el método get, que devuelve todas las columnas de la tabla, el código de la función es el siguiente:

```php
public function get_biblioteca(){
    $query = $this->db->get('libros');
    return $query->result_array();
}
```

Ya obtenido los datos y añadido la vista al controlador, lo único que hace falta es crear el archivo index.php.

El contenido del archivo debe escribir una tabla obteniendo los datos de $libros cuyo valor es un array que contiene los resultados de la consulta hecha anteriormente.
Recorre el array con un for each hasta que se acaben y, escribe por columna todos los datos del libro.
El código del archivo es el siguiente:

```php
<?php

    echo '<br/><table class="table table-striped table-hover table-condesed">
        <tr>
            <th>ISBN</th>
            <th>Titulo</th>
            <th>Autor</th>
            <th>paginas</th>
            <th>CDU</th>
            <th>año publicacion</th>
        </tr>';
        foreach ($libros as $libros_item):
        echo'<tr>
                <td>' . $libros_item['ISBN'].'</td>
                <td>' . $libros_item['titulo'] . '</td>
                <td>' . $libros_item['autor'] . '</td>
                <td>' . $libros_item['paginas'] . '</td>
                <td>' . $libros_item['CDU'] . '</td>
                <td>' . $libros_item['fecha_publicacion'] . '</td>
            </tr>';

        ;
        endforeach;
    ?>
</table>
```

Vista index.php localizado en applications/views/biblioteca/

Escribe una tabla y, obteniendo los datos de $libros cuyo valor es un array que contiene los resultados de la consulta “Select * from libros”. Recorre el array con un for each hasta que se acaben y, escribe por columna todos los datos del libro.
El headerAdmin contiene código, mayoritariamente en bootstrap que muestra un menú como cabecera de la aplicación web.
Si nos logeamos correctamente, nos mostrara la siguiente página vacía si no hay libros o, si hay libros en la biblioteca con su información, será algo parecido a esto:

Vista de biblioteca/index

Conforme mas libros se añadan a la aplicación, mas filas habrá.

Buscar y modificar datos de una fila en la base de datos

Para editar los datos de un libro, el cambio principal es crear un formulario en vez de una tabla, para poder modificar los datos y buscar qué libro queremos editar.
Empezamos añadiendo las vistas al controlador, en nuestro caso, la vista de llamará editarLibro, con lo cual, creamos la función y añadimos las vistas.

```
public function editarLibro(){
    $data['title'] = 'Editar libros';
        $this->load->view('templates/headerAdmin',$data);
        $this->load->view('biblioteca/editarLibro',$data);
        $this->load->view('templates/footer',$data);
    }
```

Función editarLibro, dentro de biblioteca.php (application/controllers).

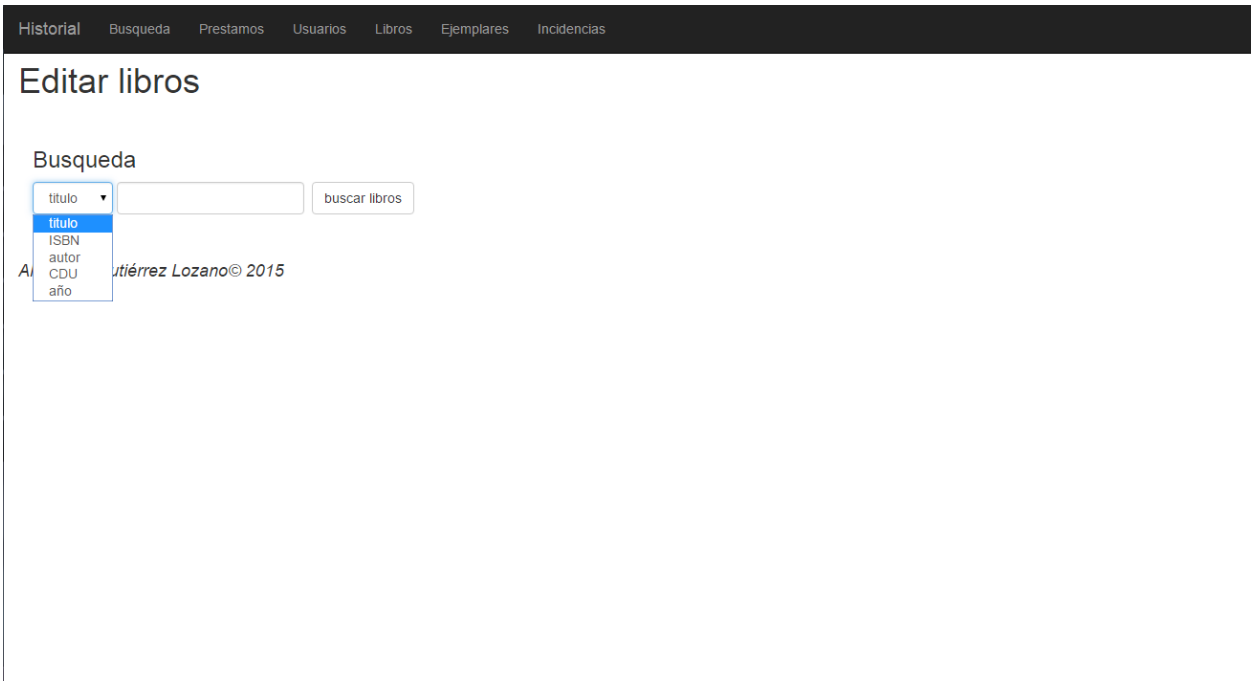
Ahora, dentro de editarLibro tenemos que hacer dos estructuras, una estructura donde recoja la información que se desea de un libro para saber qué libro debemos mostrar, pasar la información del libro a la vista y, la segunda estructura que muestre, en caso de existir el libro, su información para poder editarla.
Para la primera estructura crearemos un sencillo sistema de búsqueda donde el usuario pueda buscar por términos exactos algún dato del libro.
Para ello, vamos a crear un formulario con dos campos, una lista con los atributos por los que buscar el libro y, un campo de texto.

El código es el siguiente:

```php
<?php

echo validation_errors();

$attributes = array(
    'class' => 'form-inline',
    );
$clase = array(
    'class' => 'form-control'
    );
$opciones = array(
    'titulo' => 'titulo',
    'ISBN' => 'ISBN',
    'autor' => 'autor',
    'cdu' => 'CDU',
    'fecha_publicacion' => 'año',
    );
$dato = array(
    'class' => 'form-control',
    'name' => 'dato',
    );
echo '<div class="container-fluid">';
echo
    form_open('biblioteca/editarLibro',$attributes).
    '<br/><h3>Busqueda</h3>
            '.form_dropdown('tipo',$opciones,'',$clase).'
        <div class="form-group">'.
            form_input($dato)
        .'<br/></div>
    <input class="btn btn-default" type="submit" name="buscar" value="buscar libros"></form><br/>';
```

Archivo editarLibro.php (application/views/biblioteca)

Creamos un formulario con form_open, mencionado anteriormente, un inputs y una función llamada dropdown que crea una lista desplegable. Tiene cuatro parámetros siendo el último opcional. El primero es el nombre de la lista, el segundo las opciones, el tercero si se desea que alguna opción esté elegida por defecto y, el cuarto parámetro si se quiere añadir algo al select, en mi caso añadirle la clase form-control.
Una vez hecho, si accedemos a http://localhost/Biblioteca/editarLibro la página nos debe salir algo parecido a esto:



Página biblioteca/editarLibro

Una vez hecho, el siguiente paso es comprobar si existe un libro con el conjunto de datos que hemos introducido. Para ello, en el controlador debemos obtener el valor escogido de la lista y el valor introducido en el campo de texto.
El código es el siguiente:

```php
public function editarLibro(){
    $data['title'] = 'Editar libros';
    if($this->input->post('buscar')){
        $dato = $this->input->post('dato');
        $tipoBusqueda = $this->input->post('tipo');
        $data['libro'] = $this->biblioteca_model->get_libro($dato,$tipoBusqueda);
    }
    else{
        $this->load->view('templates/headerAdmin',$data);
        $this->load->view('biblioteca/editarLibro',$data);
        $this->load->view('templates/footer',$data);
    }
```

Función editarLibro en biblioteca.php

‘buscar’ es el nombre del botón de submit. $dato tiene como valor lo introducido en el campo de texto. $tipoBusqueda es la opción escogida de la lista.
Una vez obtenido los datos, debemos buscar si hay alguna coincidencia en la base de datos, para ello, nos vamos a biblioteca_model.php y añadimos una nueva función que devuelva esa consulta.

```php
public function get_libro($dato,$tipoBusqueda){
    $this->db->where($tipoBusqueda,$dato);
    $query = $this->db->get('libros');
    return $query->result_array();
}
```

Función get_libro de biblioteca_model (application/models)

La función where tiene dos parámetros como se puede observar, el primero indica qué campo de la tabla quieres buscar. El segundo es el valor que quieres buscar.
Por ejemplo, si quiero buscar cuyo ISBN es 978-84-16047-34-5, sería $this->db->where('ISBN',' 978-84-16047-34-5');

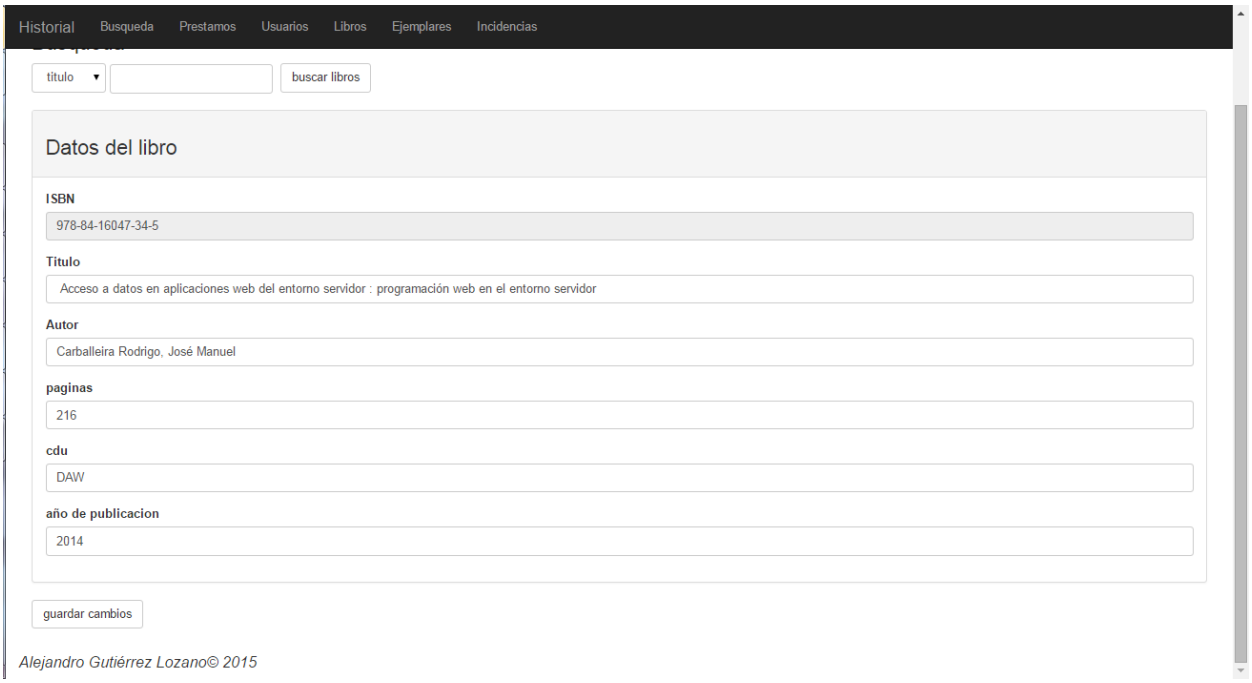
Ahora, debemos añadir los datos del libro a la vista si existe o, en otro caso, mostrar de nuevo la búsqueda.
Nos vamos a la vista y debemos comprobar que, si la consulta tiene un valor, es decir, si no está vacía, mostrar los datos del libro en un formulario para que se puedan editar. Dicha condición se realiza con la función empty de PHP.

El código del formulario es el siguiente:

```php
if(!empty($libro)){
    echo form_open('biblioteca/editarLibro');

    foreach ($libro as $libro_item):

    if(empty($isbn)){
        $isbn = $libro_item['ISBN'];
        $titulo = $libro_item['titulo'];
        $autor = $libro_item['autor'];
        $paginas = $libro_item['paginas'];
        $cdu = $libro_item['CDU'];
        $fecha_publicacion = $libro_item['fecha_publicacion'];
        $hidden = array('isbn' => $isbn);
        echo form_hidden($hidden).
        '    <div class="panel panel-default">
                <div class="panel-heading"><h3>Datos del libro</h3></div>
                <div class="panel-body">
                <div class="form-group">
                        <label>ISBN</label>
                        <input type="name" maxlength="120" class="form-control" value="'.$isbn.'" disabled>
                    </div>
                    <div class="form-group">
                        <label>Titulo</label>
                        <input type="name" name="titulo"  maxlength="255" class="form-control" value="'.$titulo.'" required>
                    </div>
                    <div class="form-group">
                        <label>Autor</label>
                        <input type="name" name="autor" maxlength="120" class="form-control" value="'.$autor.'" required>
                    </div>
                    <div class="form-group">
                        <label>paginas</label>
                        <input type="number" name="paginas" maxlength="11" class="form-control" value="'.$paginas.'">
                    </div>
                    <div class="form-group">
                        <label>cdu</label>
                        <input type="name" name="cdu" maxlength="120" class="form-control" value="'.$cdu.'">
                    </div>
                    <div class="form-group">
                        <label>año de publicacion</label>
                        <input type="number" name="fecha_p"  maxlength="11" class="form-control" value="'.$fecha_publicacion.'">
                    </div>
                </div>
            </div>';
    }
    endforeach;
    }
    if(!empty($libro)){
            echo '<input class="btn btn-default" type="submit" name="editar" value="guardar cambios">';
        }
```

Archivo editarLibro.php en (application/views/biblioteca)

Obtenemos los datos del libro y creamos un formulario con esos datos y un botón para que los cambios realizados se actualicen en la base de datos, además de un input oculto con el valor del ISBN que usaremos más adelante.
Si esta todo correcto, al introducir una búsqueda correcta nos debe salir algo parecido a esta página.



Vista de la pagina biblioteca/editarLibro.php con formulario para editar el libro.

Una vez editado y modificado los datos que deseamos, al pulsar guardar cambios, debemos recoger los datos del formulario, enviarlos al controlador y de este al modelo de datos (biblioteca_model) para que modifique el libro.
Para recoger los datos, debemos comprobar que el botón ha sido pulsado y recoger los datos.
Nos vamos al controlador y añadimos lo siguiente a la función editarLibro.

```
if($this->input->post('editar')){
    $isbn = $this->input->post('isbn');
    $nuevoTitulo = $this->input->post('titulo');
    $nuevoAutor = $this->input->post('autor');
    $nuevoPaginas = $this->input->post('paginas');
    $nuevocdu = $this->input->post('cdu');
    $nuevoFecha = $this->input->post('fecha_p');
    $data['title'] = 'Libros';
    $this->biblioteca_model->set_libro($isbn,$nuevoTitulo,$nuevoAutor,$nuevoPaginas,$nuevocdu,$nuevoFecha);
    $this->biblioteca_model->add_historial('Editado el libro cuyo ISBN era: <b>'.$isbn.'</b>');
    $data['libros'] = $this->biblioteca_model->get_biblioteca();
    $this->load->view('templates/headerAdmin',$data);
    $this->load->view('biblioteca/index',$data);
    $this->load->view('templates/footer',$data);
}
```

Quedándose finalmente con este código:

```php
public function editarLibro(){
    $data['title'] = 'Editar libros';
    if($this->input->post('buscar')){
        $dato = $this->input->post('dato');
        $tipoBusqueda = $this->input->post('tipo');
        $data['libro'] = $this->biblioteca_model->get_libro($dato,$tipoBusqueda);
    }
    //Si se guardan los datos
    if($this->input->post('editar')){
        $isbn = $this->input->post('isbn');
        $nuevoTitulo = $this->input->post('titulo');
        $nuevoAutor = $this->input->post('autor');
        $nuevoPaginas = $this->input->post('paginas');
        $nuevocdu = $this->input->post('cdu');
        $nuevoFecha = $this->input->post('fecha_p');
        $data['title'] = 'Libros';
        $this->biblioteca_model->set_libro($isbn,$nuevoTitulo,$nuevoAutor,$nuevoPaginas,$nuevocdu,$nuevoFecha);
        $data['libros'] = $this->biblioteca_model->get_biblioteca();
        $this->load->view('templates/headerAdmin',$data);
        $this->load->view('biblioteca/index',$data);
        $this->load->view('templates/footer',$data);
    }
    else{
        $this->load->view('templates/headerAdmin',$data);
        $this->load->view('biblioteca/editarLibro',$data);
        $this->load->view('templates/footer',$data);
    }
}
```

Función editarLibro en biblioteca.php

Por último, debemos crear la función set_libro en biblioteca_controller. Dicha función modificará el libro cuyo ISBN se encuentre en la tabla, la consulta es “update libros where ISBN = libro_isbn SET titulo = ‘{‘titulo’}…
El código resultante sería este:

```php
public function set_libro($isbn,$titulo,$autor,$paginas,$cdu,$fecha_p){
    $datosNuevos= array(
        'ISBN' => $isbn,
        'titulo' => $titulo,
        'autor' => $autor,
        'paginas' => $paginas,
        'cdu' => $cdu,
        'fecha_publicacion' => $fecha_p,
        );
    $this->db->where('ISBN',$isbn);
    $this->db->update('libros',$datosNuevos);
}
```

Función set_libro de biblioteca_model.

Una vez hecho, si todo esta correcto, podremos editar un libro existente en la base de datos y, si modificamos algún dato, debe aparecer ese dato modificado correctamente en la lista de libros, la cual sale al guardar los cambios.

Eliminar una fila en la base de datos

Para eliminar un libro, lo único a modificar respecto a editar es mostrar los datos del libro pero, sin poder editarlos y un mensaje que indique que esté seguro por seguridad.
La búsqueda ha sido realizada anteriormente, por lo que, no entraré en detalles.
El archivo se llama eliminarLibro y estará, como los demás, en application/views/biblioteca.
Así pues, creamos la vista y los métodos necesarios para que funcione. Es útil usar la misma función de búsqueda usada que en editarEjemplares.
Los códigos de eliminarEjemplares, y biblioteca correspondientes a esta parte son los siguientes:

```php
<?php

echo validation_errors();

$attributes = array(
    'class' => 'form-inline',
    );
echo '<div class="container-fluid">';
echo form_open('biblioteca/eliminarLibro',$attributes);
    echo '<br/><h3>Busqueda</h3>
        <select class="form-control" name="tipo">
            <option value="titulo">titulo</option>
            <option value="ISBN">ISBN</option>
            <option value="autor">autor</option>
            <option value="CDU">CDU</option>
            <option value="fecha_publicacion">año</option>
        </select>
        <div class="form-group">
            <input class="form-control" name="dato" type="text" value=""><br/>
        </div>
    <input class="btn btn-default" type="submit" name="buscar" value="buscar libros"></form><br/>';
```

eliminarLibro.php

```php
public function eliminarLibro(){
    $data['title'] = 'Eliminar libros';
    if($this->input->post('buscar')){
        $dato = $this->input->post('dato');
        $tipoBusqueda = $this->input->post('tipo');
        $data['libro'] = $this->biblioteca_model->get_libro($dato,$tipoBusqueda);
    }
    $this->load->view('templates/headerAdmin',$data);
    $this->load->view('biblioteca/eliminarLibro',$data);
    $this->load->view('templates/footer',$data);
}
```

Función eliminarLibro en biblioteca.php

Con la opción **disabled** dentro de los atributos del input no permitimos que se puedan editar. Necesitaremos el ISBN así que creamos un input oculto con su valor.
El código mostrando los datos y el mensaje es el siguiente:

```php
foreach ($libro as $libro_item):

if(empty($isbn)){
    $isbn = $libro_item['ISBN'];
    $titulo = $libro_item['titulo'];
    $autor = $libro_item['autor'];
    $paginas = $libro_item['paginas'];
    $cdu = $libro_item['CDU'];
    $fecha_publicacion = $libro_item['fecha_publicacion'];
    $hidden = array('isbn' => $isbn);
    echo form_hidden($hidden).
    '    <div class="panel panel-default">
            <div class="panel-heading"><h3>Datos del libro</h3></div>
            <div class="panel-body">
            <div class="form-group">
                    <label>ISBN</label>
                    <input type="name" class="form-control" value="'.$isbn.'" disabled>
                </div>
                <div class="form-group">
                    <label>Titulo</label>
                    <input type="name" class="form-control" value="'.$titulo.'"disabled>
                </div>
                <div class="form-group">
                    <label>Autor</label>
                    <input type="name" class="form-control" value="'.$autor.'"disabled>
                </div>
                <div class="form-group">
                    <label>paginas</label>
                    <input type="name" class="form-control" value="'.$paginas.'"disabled>
                </div>
                <div class="form-group">
                    <label>CDU</label>
                    <input type="name" class="form-control" value="'.$cdu.'"disabled>
                </div>
                <div class="form-group">
                    <label>año de publicacion</label>
                    <input type="name" class="form-control" value="'.$fecha_publicacion.'"disabled>
                </div>';
            if(!empty($libro)){
        echo '<br/><b>Estas seguro que quieres eliminar este libro</b>
            <input style="border-radius:5px; margin: 5px; padding:0px 10px;" type="submit" name="eliminar" value="Si">
            <input style="border-radius:5px; margin: 5px; padding:0px 10px;" type="submit" name="salir" value="No">';
    }
        echo '</div>
            </div>
        </div>';
}
endforeach;
```

Código de eliminarLibro.php

Una vez realizado, al intentar eliminar un libro nos debe salir una vista parecida a esta en el navegador

Vista eliminarLibro en el navegador

Al darle si, debemos eliminar el libro y, con ello, todos los ejemplares, en caso contrario, nos devolvería de nuevo al principio.
Debemos añadir un nombre al botón **si** para que cuando sea pulsado, elimine el libro. En mi caso, el nombre es eliminar como se puede observar en el código.
Al darle a unos de los botones debemos comprobar si dicho botón ha sido pulsado y, eliminar el libro junto con sus ejemplares.
El código de dicha parte del controlador es:

```
if($this->input->post('eliminar')){
    $isbn = $this->input->post('isbn');
    $this->biblioteca_model->delete_libro($isbn);
}
```

Quedándose finalmente de la siguiente manera:

```php
//Funcion de la pestaña eliminar libro
public function eliminarLibro(){
    $data['title'] = 'Eliminar libros';
    if($this->input->post('buscar')){
        $dato = $this->input->post('dato');
        $tipoBusqueda = $this->input->post('tipo');
        $data['libro'] = $this->biblioteca_model->get_libro($dato,$tipoBusqueda);
    }
    //Si se elimina el libro
    if($this->input->post('eliminar')){
        $isbn = $this->input->post('isbn');
        $this->biblioteca_model->delete_libro($isbn);
    }
    $this->load->view('templates/headerAdmin',$data);
    $this->load->view('biblioteca/eliminarLibro',$data);
    $this->load->view('templates/footer',$data);
}
```

Función completa de eliminarLibro en biblioteca.php (application/controllers)

Solo nos queda añadir el método delete_libro($isbn) en el modelo biblioteca_model. La función para eliminar una fila en la base de datos es **delete** y necesita dos parámetros. El primero es la tabla a eliminar los datos, el segundo es una condición cuyo fin es borrar sólo aquellos que la cumplan.
El código de la función es:

```php
public function delete_libro($isbn){

    $this->db->delete('libros',array('ISBN'=>$isbn));
    $this->db->delete('ejemplares',array('libros_ISBN'=>$isbn));
}
```

Función delete_libro($isbn) en biblioteca_controller (applications/controllers)

Una vez hecho, si le damos a “si” en la vista de eliminar libros, el sistema borrará el libro y todos los ejemplares asociados al ISBN.

Unir dos tablas para obtener campos de ambas tablas

Para obtener todos los datos de los ejemplares no es obligatorio juntar dos tablas, pero deseo que se muestre el título del libro asociado al ISBN del ejemplar.
Ya hemos mencionado como se obtiene los datos de una tabla, como se crea la vista y todo lo anterior. Los tres pasos son:
crear el archivo de la vista, en este caso indexEjemplares.php dentro de applications/views/biblioteca.
Su contenido es similar a index.php, variando solamente los atributos del bucle for each y la variable que itera por ejemplares.
El código de la vista es el siguiente:

```php
<?php
$this->load->helper('form');
echo validation_errors();

echo '<br/>
            <div class="panel panel-default">
            <div class="panel-heading"><h3>'.$title.'</h3></div>
                <div class="panel-body">
                    <table class="table table-striped table-hover table-condesed">
                        <tr>
                            <th>Codigo del ejemplar</th>
                            <th>ISBN</th>
                            <th>Titulo</th>
                            <th>Estado</th>
                        </tr>';
                    foreach ($ejemplares as $ejemplares_item):
                    echo'<tr>
                            <td>' . $ejemplares_item['cod_ejemplar'].'</td>
                            <td>' . $ejemplares_item['libros_ISBN'].'</td>
                            <td>' . $ejemplares_item['titulo'] . '</td>
                            <td>' . $ejemplares_item['estado'] . '</td>
                        </tr>';
                    endforeach;
    echo '          </table>
                </div>
            </div>';
    ?>
```

Archivo indexEjemplares.php

A continuación se añade la vista en el controlador:

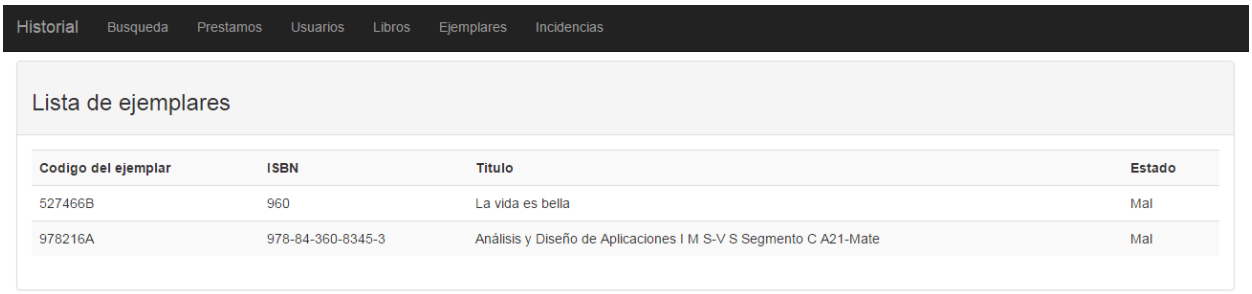
```php
public function indexEjemplares(){
    $data['ejemplares'] = $this->biblioteca_model->get_ejemplares();
    $data['title'] = "Lista de ejemplares";

    $this->load->view('templates/headerAdmin',$data);
    $this->load->view('biblioteca/indexEjemplares',$data);
    $this->load->view('templates/footer',$data);
}
```

Función indexEjemplares de biblioteca.php

Por último, se añade get_ejemplares en biblioteca_model.
Para unir dos tablas se puede usar la función join, dicha función necesita dos parámetros. La tabla a unir y, los parámetros que tienen en común dichas tablas.
En mi caso, es “libros.ISBN = ejemplares.libros_ISBN”.
El código es el siguiente:

```php
public function get_ejemplares(){
    $this->db->from('ejemplares');
    $this->db->join('libros','libros.ISBN=ejemplares.libros_ISBN');
    $query = $this->db->get();
    return $query->result_array();
}
```

Al usar la función, tenemos acceso a todos los atributos de ambas tablas.
Si lo hemos hecho todo bien e insertamos algún ejemplar, nos debe salir una tabla parecida a esta en biblioteca/indexEjemplares



Alejandro Gutiérrez Lozano© 2015

Vista de indexEjemplares en el navegador

A partir de esta implementación, el resto de las vistas han sido hechas de manera similar, variando los datos a mostrar y las operaciones hechas con los datos debido a la naturaleza de la aplicación.

7. Recursos

7.1- Herramientas hardware

Las herramientas hardware utilizadas son:
Portátil.

El ordenador que me ha permitido seguir el proyecto en varios sitios por su fácil transporte. El portátil tiene un coste de unos 600€.

Cargador de batería.

Herramienta para cargar la batería del portátil conectándolo a un enchufe. El coste fue de 20€.

Batería.

Herramienta para tener el portátil encendido sin tener que usar el cargador. Cuesta unos 30€.

Ratón.

Para trabajar con el portátil de manera más cómoda que con el touchpad integrado. El que uso costó 5€.

7.2- Herramientas software

Las herramientas software expuestas a continuación tienen licencia gratuita:

Apache server.

Servidor para las pruebas de la aplicación realizada en este proyecto.

Xampp.

Software que incorpora herramientas necesarias para crear una aplicación en el entorno servidor de una manera más fácil y cómoda.

Bootstrap.

Framework de CSS diseñado especialmente para mobile design. Para dar estilo a la aplicación web.

PHP.

Lenguaje de programación necesario para hacer la aplicación web.

Phpmyadmin.

Usado como base de datos de la aplicación.

MySQL.

Como lenguaje de base de datos

CodeIgniter.

Framework usado durante todo el proyecto.

Editor de texto Sublime.

Editor de texto usado para escribir todo el código del proyecto gracias a su interfaz de ayuda.

Google Chrome.

Navegador de Internet usado para mostrar el servidor web donde se aloja la aplicación.

7.3- Personal

El personal que ha ayudado a la realización del proyecto:

Alejandro Gutiérrez Lozano. Ha hecho toda la parte que concierne a la aplicación web y documentación del proyecto.
Almudena Ortega Rodríguez. Ha participado ayudando en la parte del diseño de la base de datos y la documentación.
José Aguilera Ruiz. Su función ha sido ayudar con los objetivos y requisitos iniciales que debe de cumplir el proyecto.
-Jorge Gutiérrez Lozano. Ha ayudado de forma general durante el transcurso del proyecto.

7.4- Presupuesto

El presupuesto estimado es aproximadamente el coste total del hardware y el coste material del tiempo empleado en la realización del proyecto, debido al uso de herramientas con licencia de libre uso como software. Además de una línea de Internet y la luz consumida por el portátil.

Portátil. 600€.
Cargador de batería. 20€.
Batería. 30€.
Ratón. 5€.
Tiempo empleado aproximado: 356 horas a 8€/hora = 2848€.
Total: 3503€.

8. Conclusiones

8.1- Grado de consecución de los objetivos

De acuerdo con los objetivos principales del proyecto, la aplicación ha cumplido todos los objetivos inicialmente propuestos.

8.2- Problemas encontrados

Los mayores problemas han sido el desconocimiento inicial al trabajar con el framework porque apenas he tenido grandes dificultades por la extensa documentación que aporta el framework consigo.

8.3- Futuras mejoras

Como futuras mejoras puede tener las siguientes:
Más atractivo a la vista.
Posibilidad de ver días restantes en vez de la fecha inicio y la fecha de entrega de los préstamos.
Posibilidad de editar el número máximo permitidos de préstamos de manera sencilla.

9. Bibliografía

Guia de codeigniter: https://ellislab.com/codeigniter/user-guide/
Stackoverflow: http://stackoverflow.com/
Bootstrap: http://getbootstrap.com/css/

10. Anexos

Manual de uso

1. Entrar a la aplicación
2. Gestión de libros
   - 2.1 Lista de libros
   - 2.2 Añadir libros
   - 2.3 Editar libros
   - 2.4 Eliminar libros
3. Gestión de ejemplares
   - 3.1 Lista de ejemplares
   - 3.2 Añadir ejemplares
   - 3.3 Editar ejemplares
   - 3.4 Eliminar ejemplares
4. Gestión de usuarios
   - 4.1 Lista de usuarios
   - 4.2 Añadir usuarios
   - 4.3 Editar usuarios
   - 4.4 Eliminar usuarios
   - 4.5 Administradores
5. Gestión de préstamos
   - 5.1 Lista de préstamos
   - 5.2 Añadir préstamos
   - 5.3 Editar préstamos
   - 5.4 Eliminar préstamos
6. Gestión de incidencias
   - 6.1 Lista de incidencias
   - 6.2 Abrir incidencia
   - 6.3 Eliminar incidencias
7. Búsqueda
8. Historial

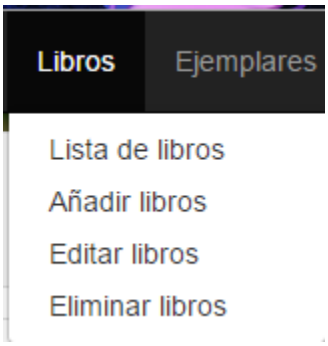
1.Entrar a la aplicación

Al entrar en la página principal del sistema nos pedirá el usuario y la contraseña, debemos introducir un usuario y contraseña válido para la aplicación, de otra forma no se podrá acceder a la aplicación de la biblioteca.

Arriba se inserta el usuario.
Abajo se inserta la contraseña del usuario.
Si los datos insertados son correctos, se entrará al sistema.
Una vez dentro, se mostrará la página principal de la aplicación que consiste en un menú superior y un listado de libros.
Dentro del menú están todas las opciones para gestionar la aplicación que se detallan a continuación:

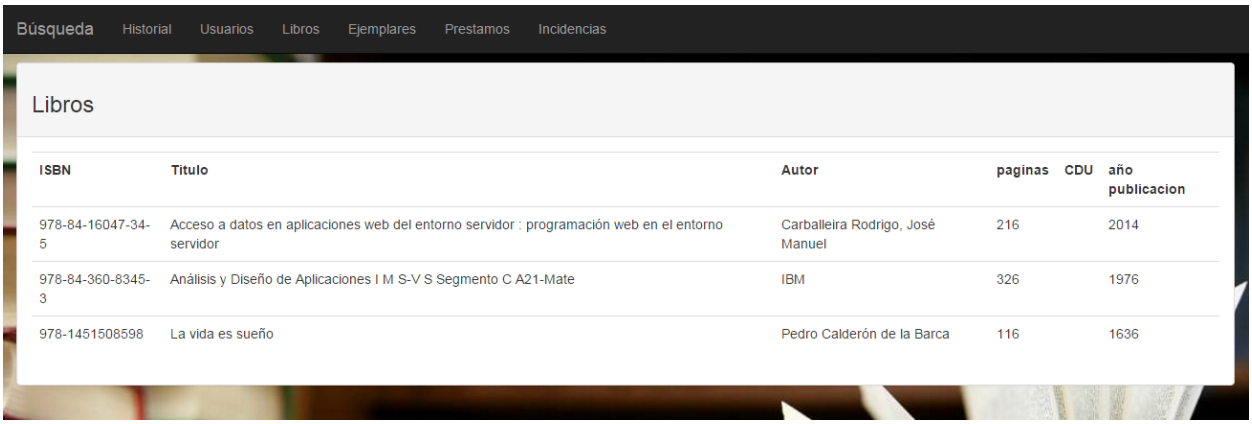
2.Gestión de libros

Dentro de la pestaña de libros en el menú superior tenemos las siguientes opciones:



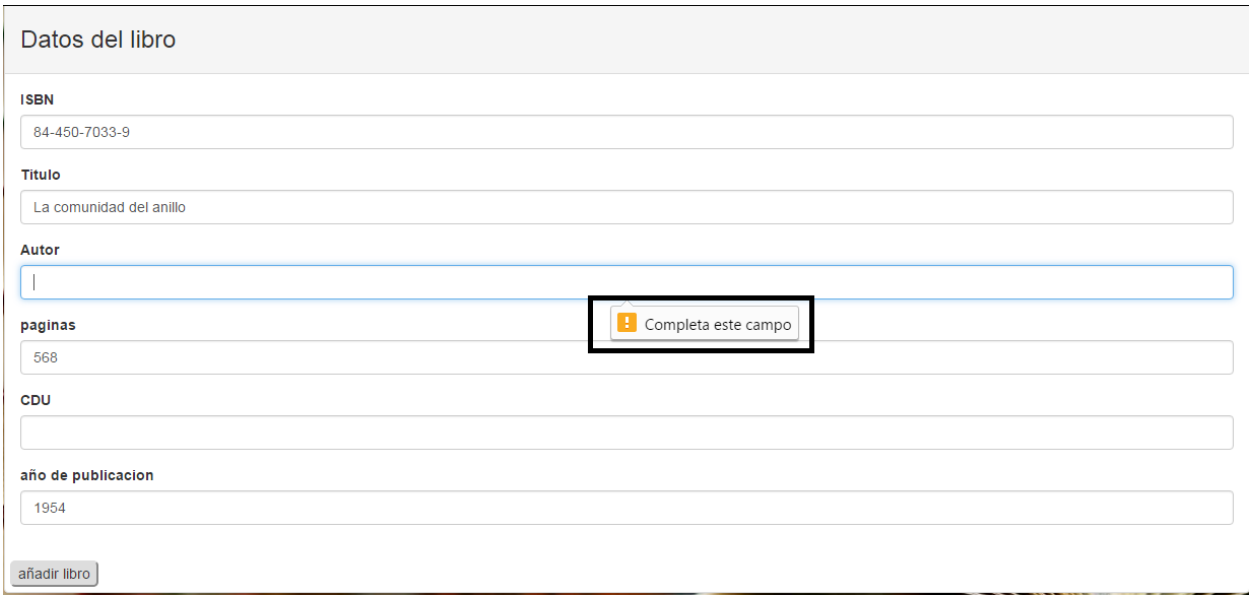
2.1 Lista de libros

Muestra un listado de todos los libros insertados en el sistema. Los libros se muestran ordenados por título.



2.2 Añadir libros

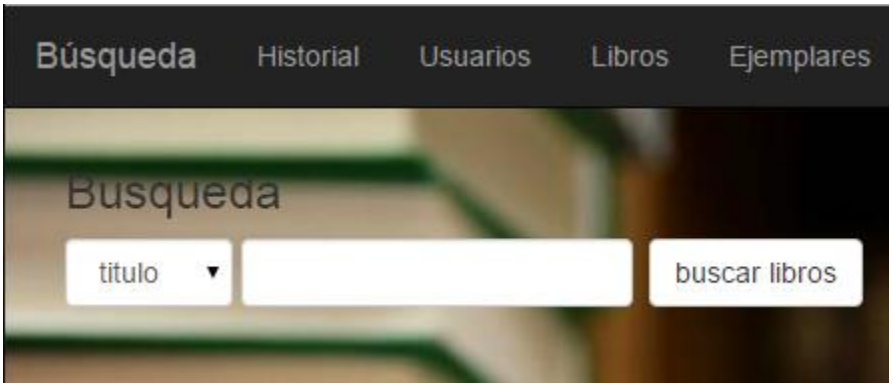
Permite añadir nuevos libros al sistema.
El ISBN, el título y el autor son campos obligatorios.
Las páginas, el CDU y el año de publicación son campos opcionales.
Si se intenta añadir un libro sin rellenar al menos los campos obligatorios se mostrará un mensaje indicando que debe completar el campo vacío.



Una vez insertado el autor (J. R. R. Tolkien), el libro será añadido al sistema.

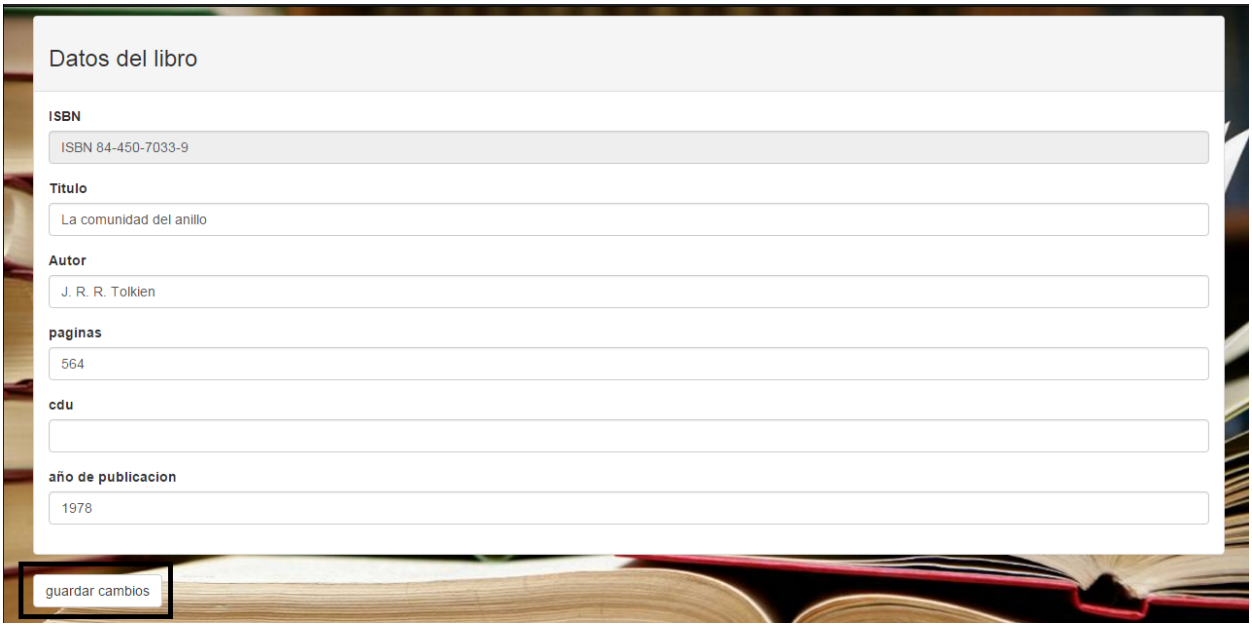
2.3 Editar libros

Nos saldrá esta página:



La página da la opción de buscar por cualquier campo y solo acepta nombres exactos, de otro modo seguirá apareciendo esta búsqueda.

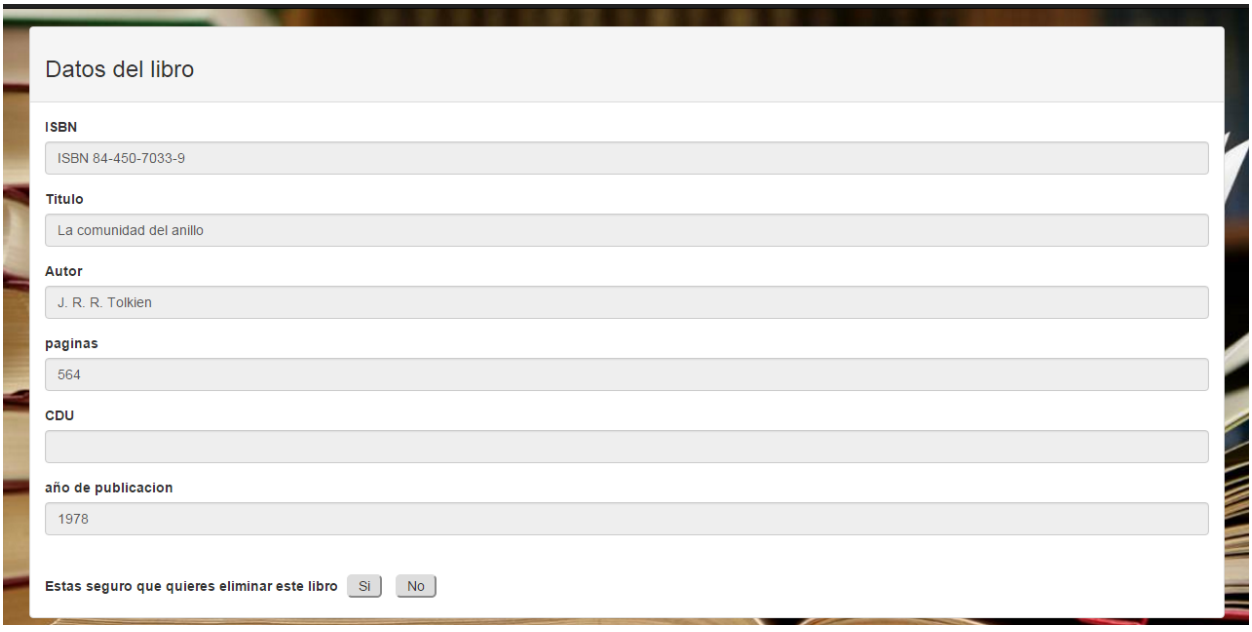
Una vez insertado un libro, nos mostrara los datos del libro y la opción a modificar todos ellos salvo el ISBN.



Si queremos guardar cambios deberemos darle al botón que se encuentra en la parte inferior del formulario. Al darle al botón nos saldrá directamente la lista de libros junto con alguna modificación si la hemos hecho.

2.4 Eliminar libros

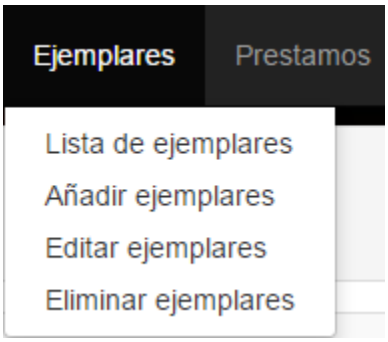
Nos muestra la misma página de búsqueda que editar libros.
Una vez insertado un nombre, nos aparecerán los datos del libro.



Si le damos a **si** eliminará el libro y **todos sus ejemplares.**
Si le damos a **no** volverá a la página de búsqueda anterior.

3.Gestión de ejemplares

Dentro de la pestaña de ejemplares tenemos las siguientes opciones:

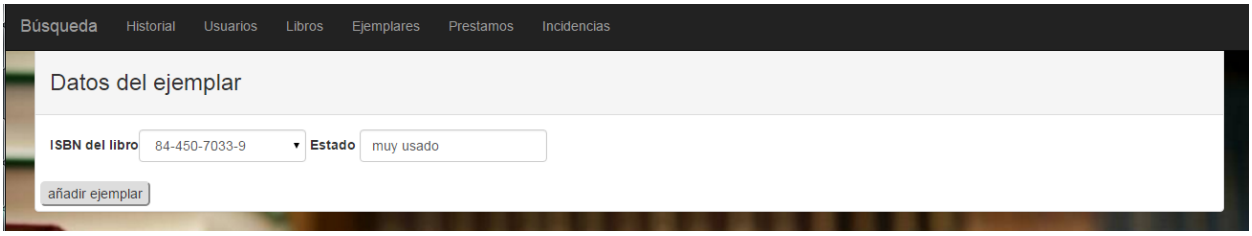


3.1 Lista de ejemplares

Muestra una lista de todos los ejemplares insertados en el sistema ordenados por nombre.

3.2 añadir ejemplares

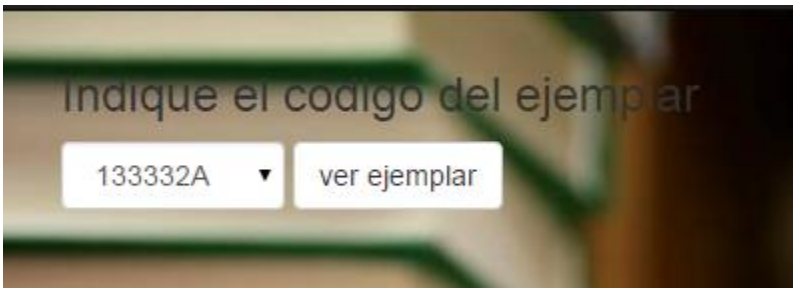
Añade nuevos ejemplares, se le debe indicar el **ISBN** del libro y el estado de conservación del ejemplar.
El código del ejemplar es generado automáticamente por el sistema.



Una vez insertado, el ejemplar es visible en la lista de ejemplares.

3.3 editar ejemplares

Permite editar ejemplares, solo deberemos indicar el código del ejemplar que queremos editar.

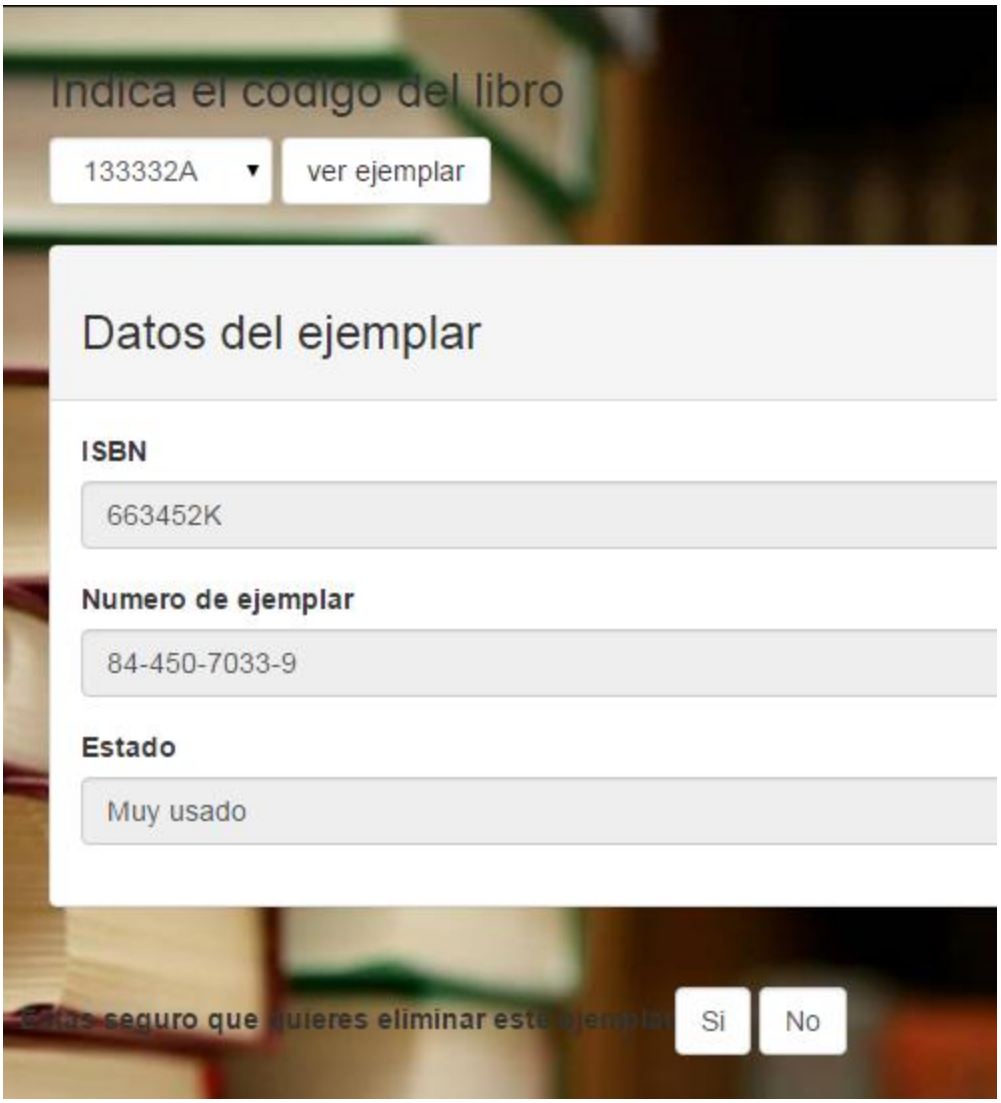


Nos mostrará los datos del ejemplar; lo único que se puede editar es el estado.

Si le damos al botón, guardaremos el nuevo estado que indiquemos.

3.4 eliminar ejemplares

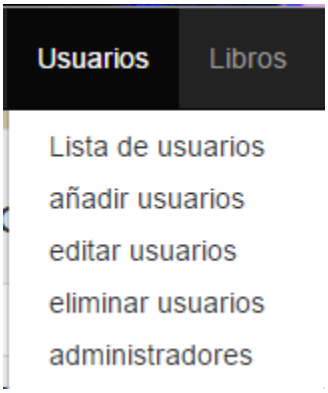
Nos permite eliminar ejemplares. Como en editar ejemplares, la búsqueda se lleva a cabo indicando el código del ejemplar.



Si le indicamos **si** borrará el ejemplar del sistema.
Si le indicamos **no** volverá a mostrar la búsqueda.

4. Gestión de usuarios

Dentro de usuarios hay las siguientes opciones:

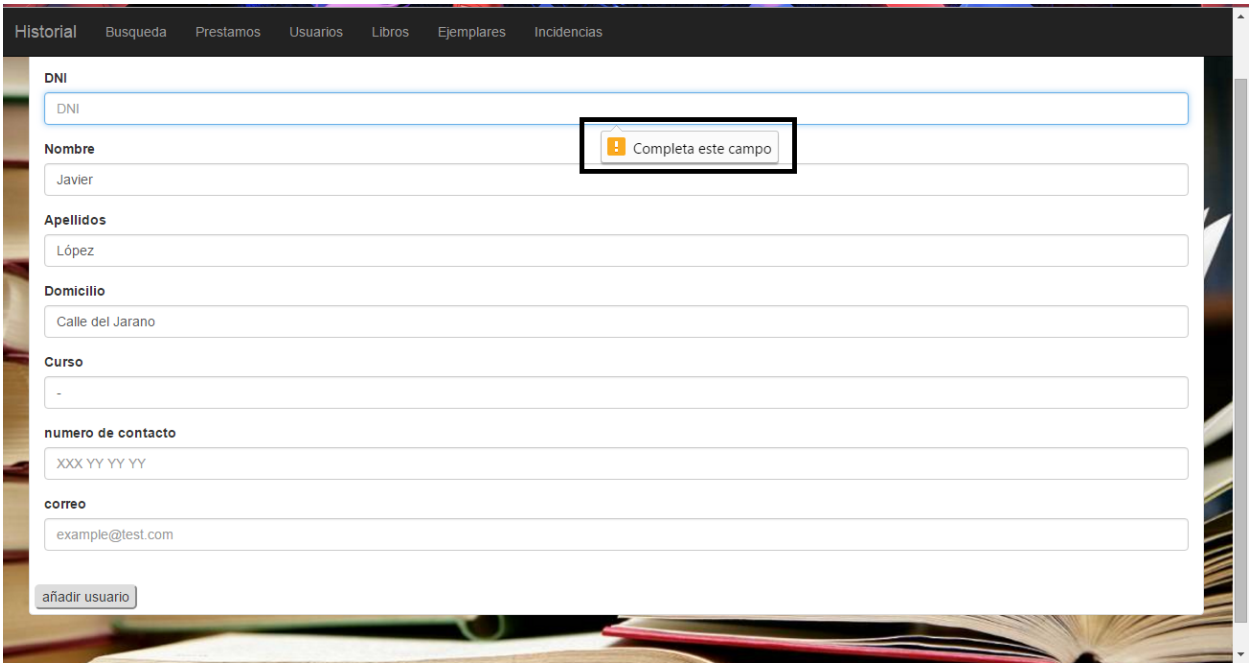


4.1 Lista de usuarios

Lista de usuarios muestra la lista de usuarios introducidos en la aplicación. Se ordena automáticamente por nombre.

4.2 Añadir usuarios

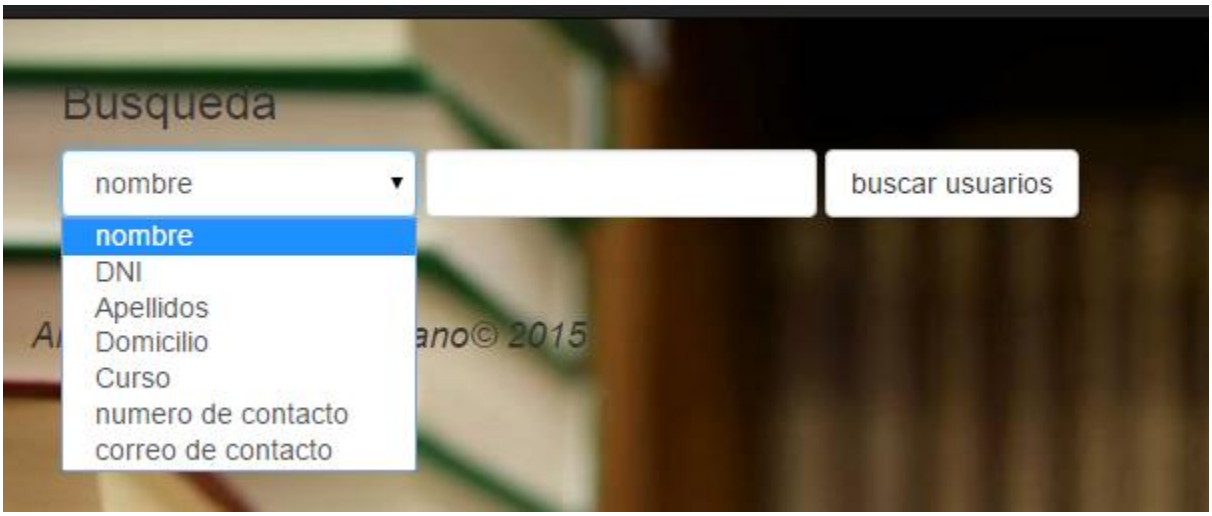
Permite añadir nuevos usuarios al sistema, todos los campos son requeridos para crear un nuevo usuario. En caso de intentar crear un usuario sin rellenar todos los campos nos saltará un aviso



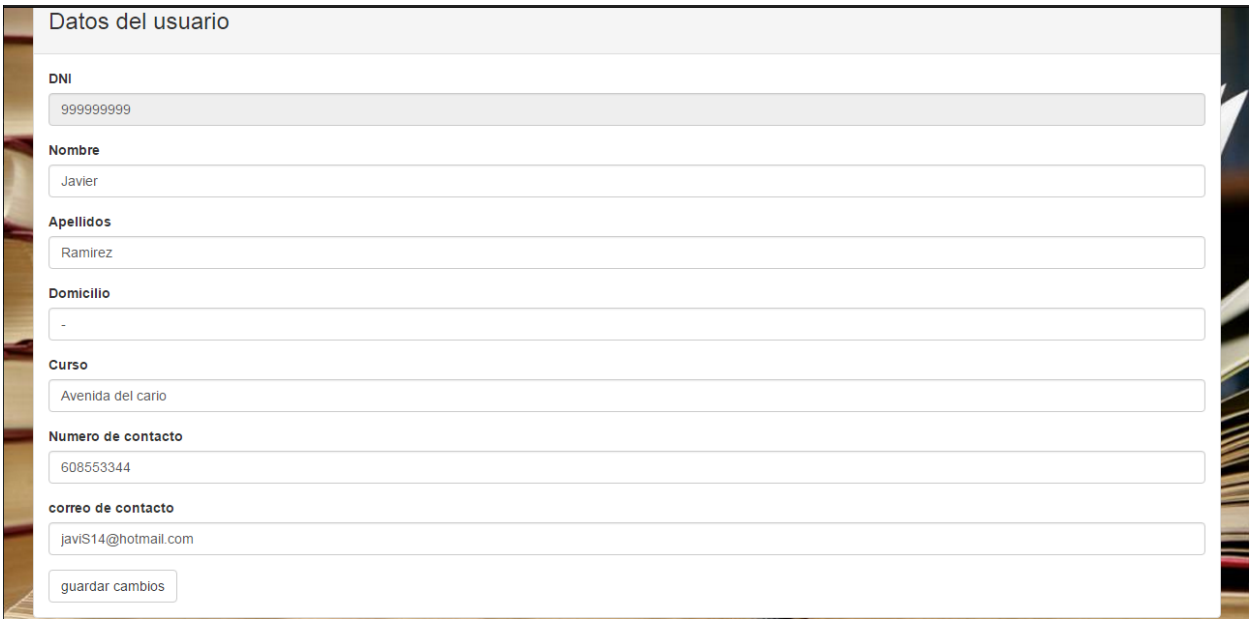
Una vez rellenado todos los campos, se insertará al usuario en el sistema.

4.3 Editar usuarios

Editar usuarios ofrece la posibilidad de buscar por los distintos datos de los usuarios. La búsqueda debe ser por términos exactos, de otro modo no habrá ningún resultado.



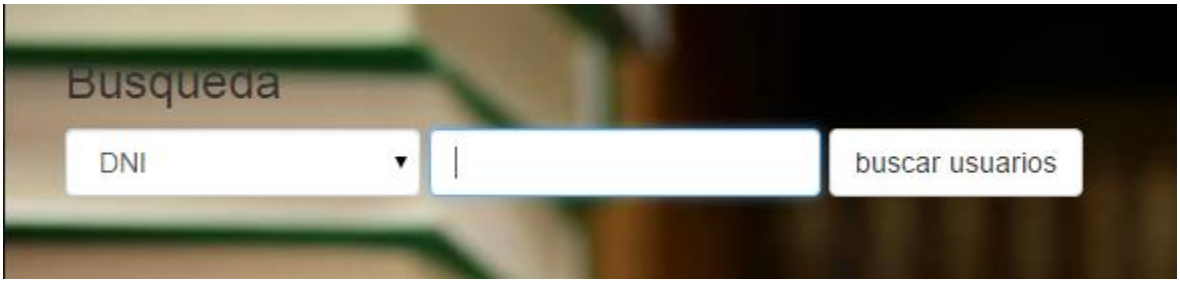
Una vez elegido un dato correcto, se puede modificar todos los datos de cualquier usuario existente salvo el DNI.



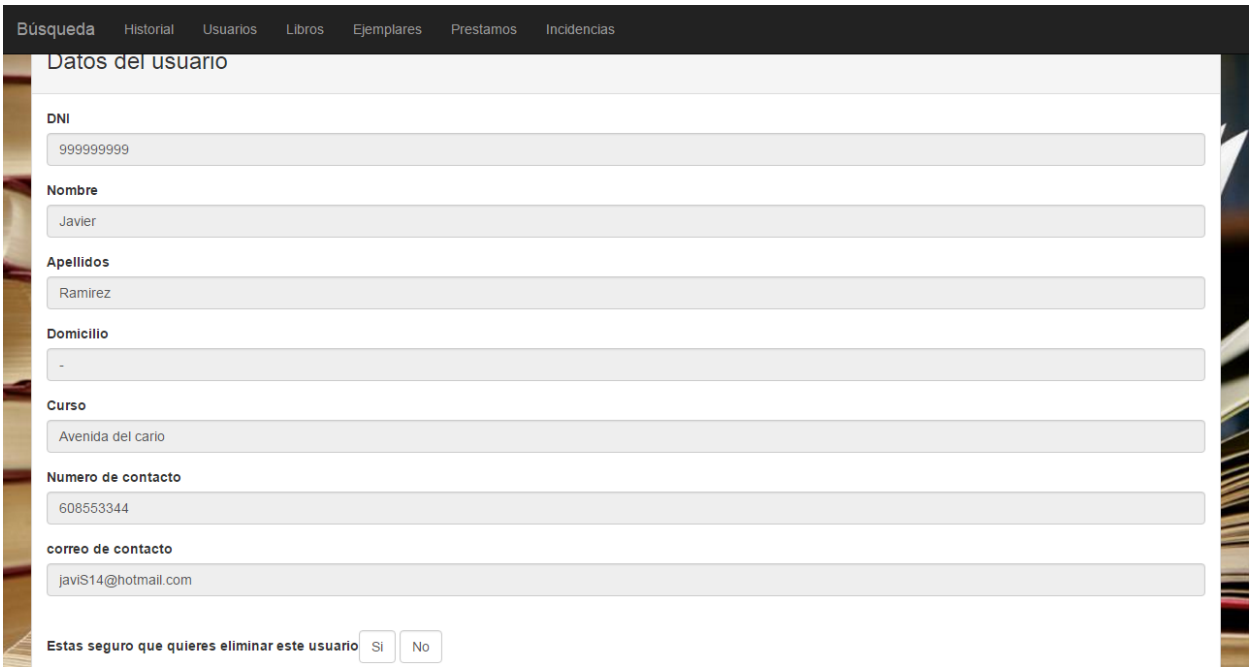
Una vez hecho las modificaciones que deseamos, le damos al botón de **guardar cambios** si queremos que se guarden dichas modificaciones.

4.4 Eliminar usuarios

Eliminar usuarios, permite la búsqueda por cualquier campo, como en editar usuarios.



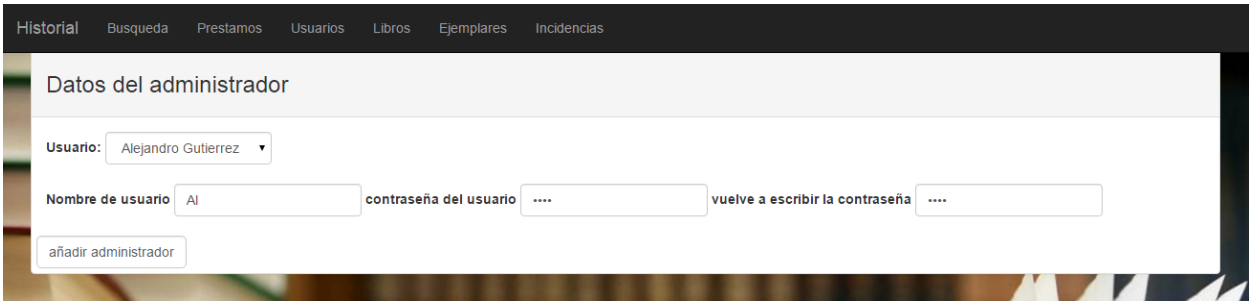
Una vez se le indica un DNI de un usuario:



Si le indicamos **si** borrará al usuario junto con **sus préstamos** y, si es administrador **su usuario/contraseña**
Si le indicamos **no** volverá a la opción de buscar.

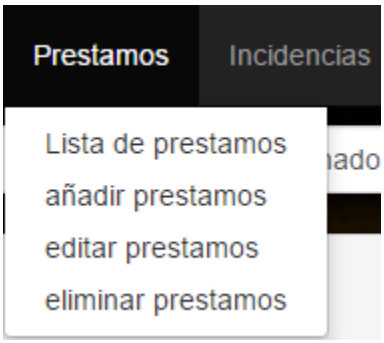
4.5 Administradores

Administradores ofrece la opción de añadir nuevos administradores a la base de datos además del que tiene la aplicación por defecto.
Se elige el usuario que va a ser administrador y se añade un nombre de usuario y contraseña que se usará para entrar al sistema.



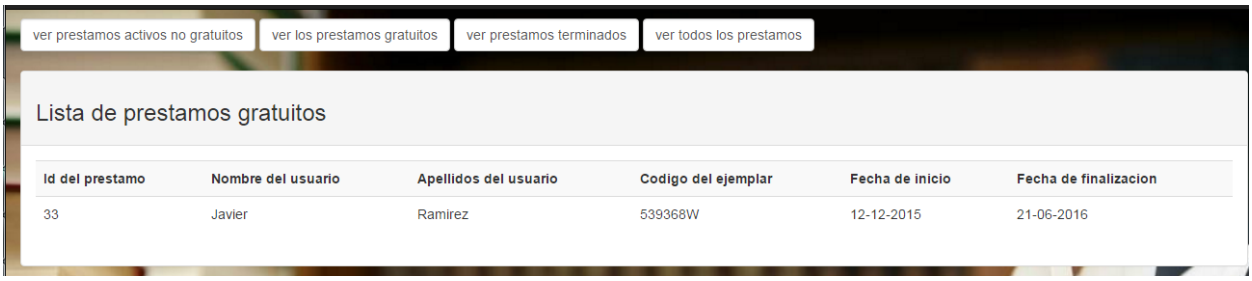
5.Préstamos

La Pestaña de préstamos manipula toda la parte de los préstamos. Tiene las siguientes opciones:



5.1 Lista de préstamos

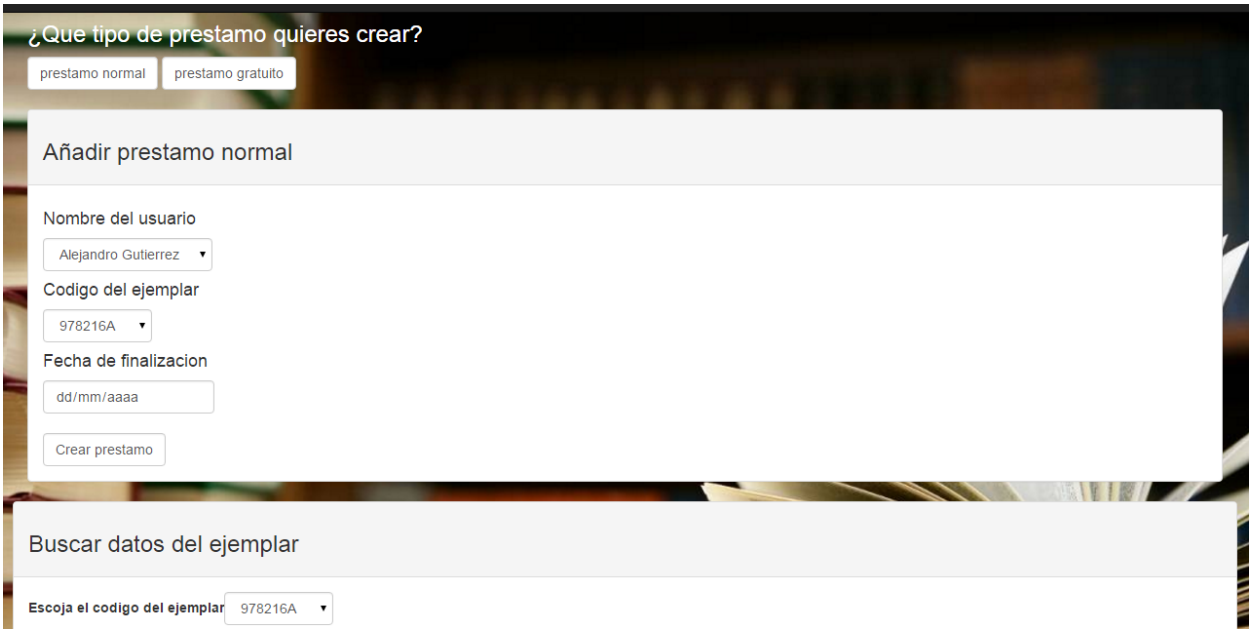
Permite ver todos los préstamos creados de manera organizada.
Préstamos activos muestra los préstamos normales.
Préstamos gratuitos muestra los préstamos gratuitos. Los préstamos gratuitos son aquellos que se prestan los libros escolares a los alumnos y los devuelven al final del curso lectivo, permitiendo reutilizarlos durante varios años.
Prestamos terminados muestra aquellos prestamos que han terminado.
Ver todos los préstamos reúne todos los préstamos. Está organizado, de manera que comienza desde los préstamos activos y termina con los préstamos terminados.



Ejemplo de la pestaña **prestamos gratuitos**

5.1 Añadir préstamos

Se mostrará esta ventana:

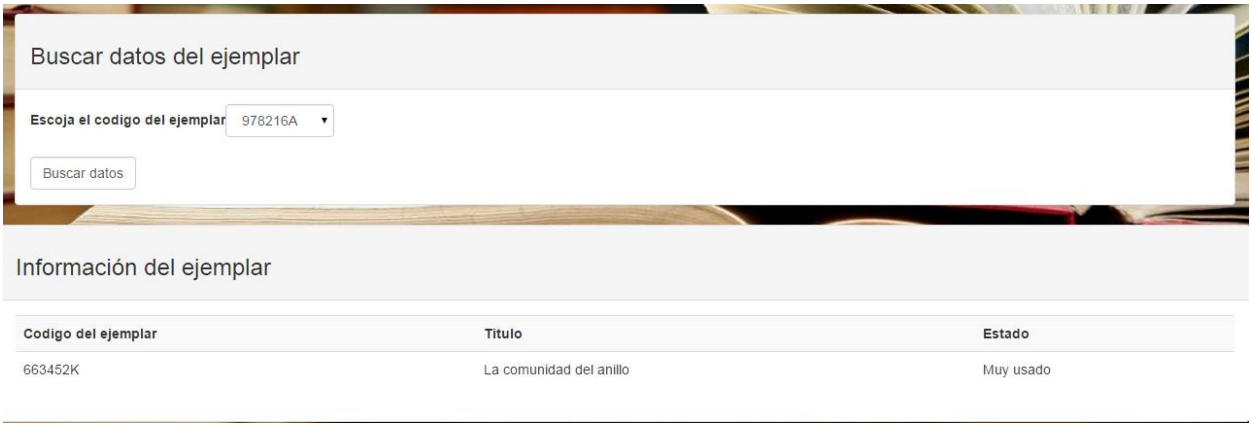


Arriba están ambas opciones para elegir el tipo de préstamo. Un préstamo normal o un préstamo gratuito. La diferencia es la fecha de finalización.

El préstamo gratuito: No se puede elegir la fecha de finalización. La fecha de finalización es el final del curso lectivo.

El préstamo normal: se puede elegir una fecha de finalización, dando la posibilidad de dejarla en blanco si así se desea.

Debajo hay una opción de buscar datos del ejemplar si queremos saber a qué libro y qué estado tiene el ejemplar.



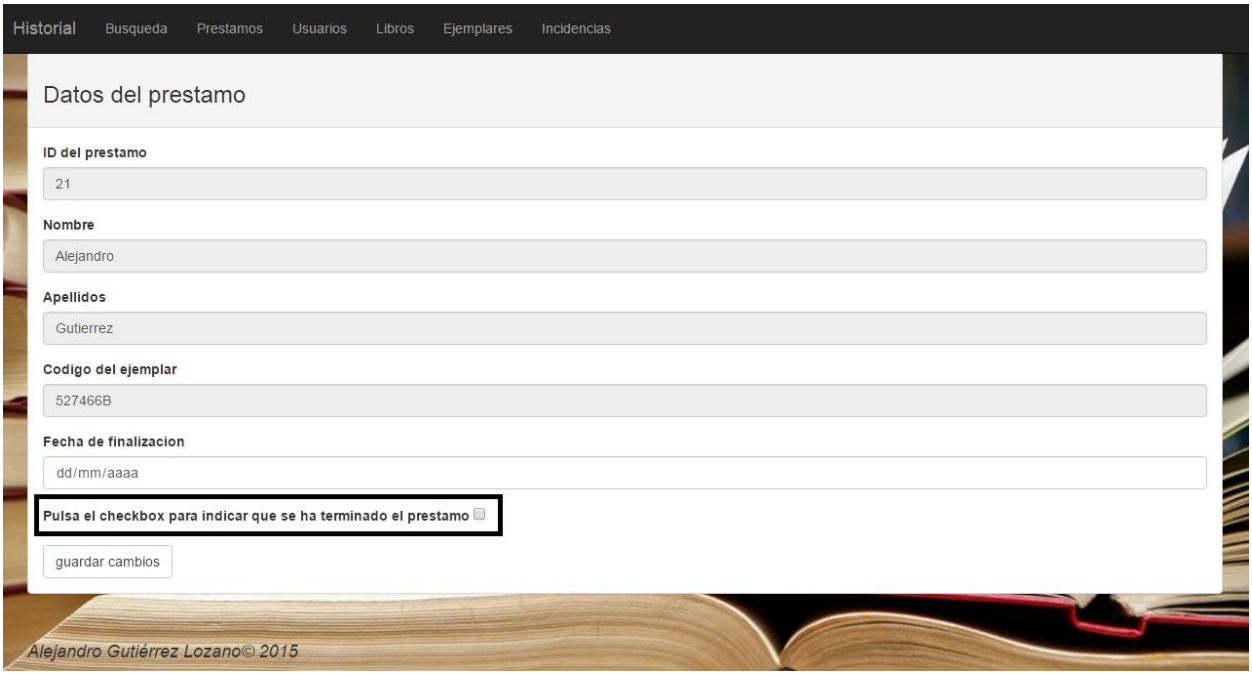
El sistema notificará un mensaje en caso de intentar realizar alguna de estas acciones no permitidas por la aplicación:

añadir un ejemplar en uso.

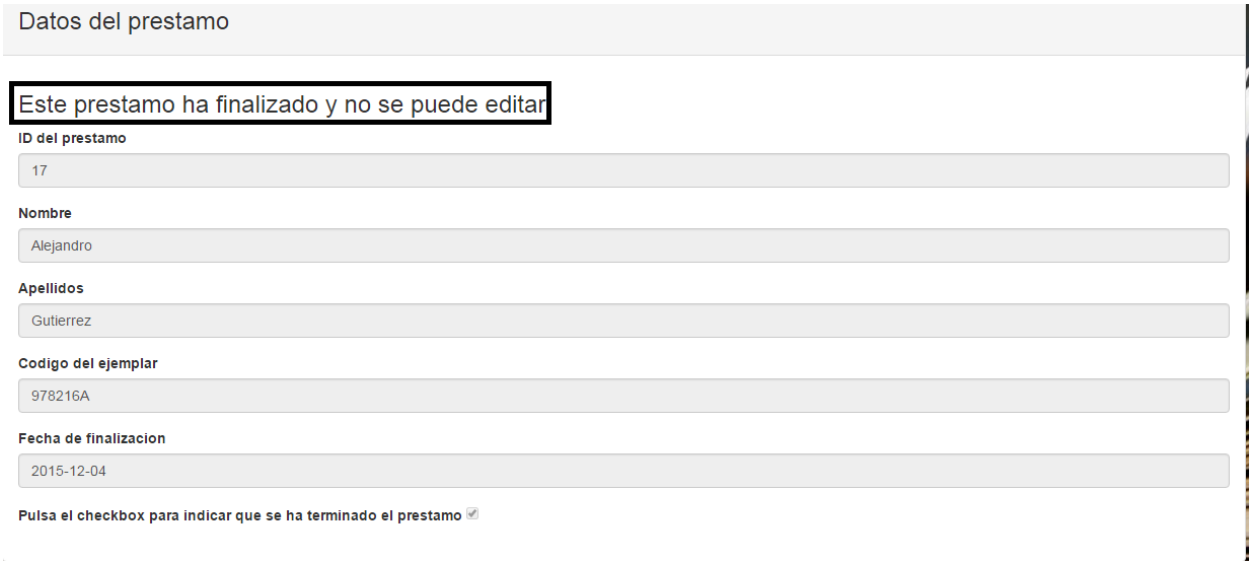
Añadir más de 4 préstamos normales simultáneos a un usuario.

5.3 editar préstamos

Editar préstamos permite la edición de préstamos. Para ello, necesitamos el id del préstamo a editar. El id es observable en la lista de préstamos.
Lo único que se puede editar es la fecha de finalización y dar por concluido el préstamo pulsando el checkbox debajo de los datos del préstamo tal y como señala la siguiente imagen:

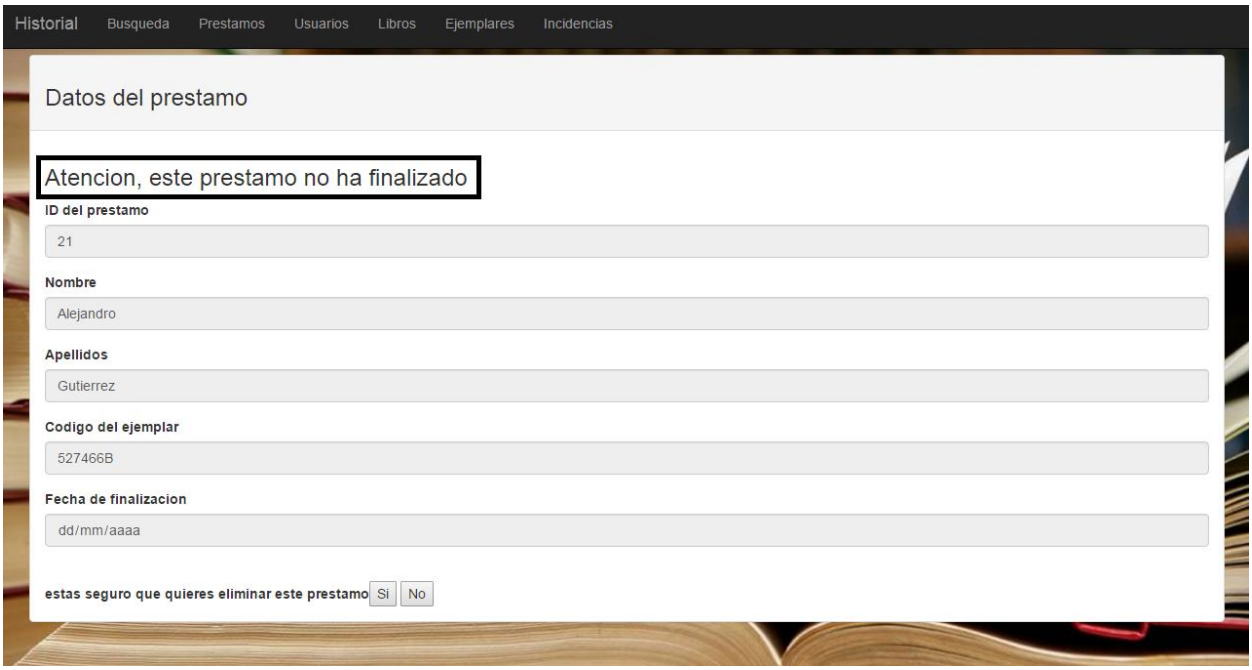


En caso de querer editar un préstamo que ya ha finalizado, nos saldrá un mensaje que nos indica que el préstamo no es editable.



5.4 Eliminar préstamos

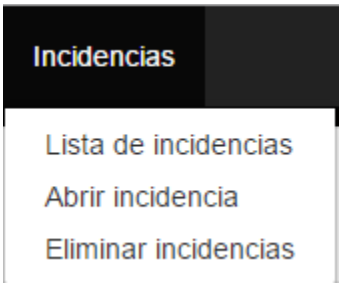
Eliminar préstamos elimina de forma permanente el préstamo del sistema.
En caso de intentar borrar un préstamo activo, nos saldrá un aviso en la parte superior de los datos del préstamo.



Una vez eliminado, el préstamo desaparece de la aplicación

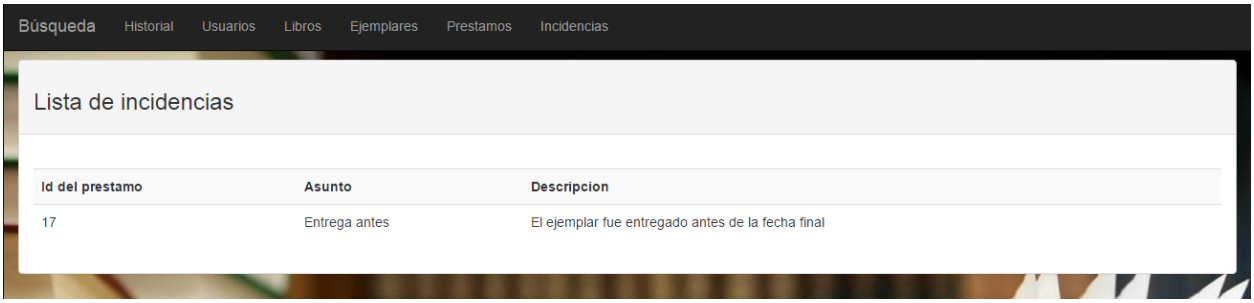
6. Incidencias

La pestaña de incidencias permite manejar las incidencias. Tiene las siguientes opciones:



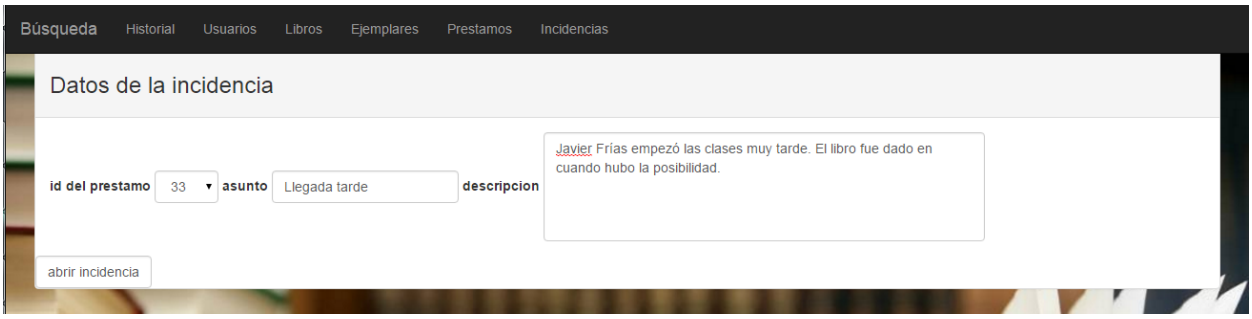
6.1 Lista de incidencias

muestra todas las incidencias que existen



6.2 Abrir incidencia

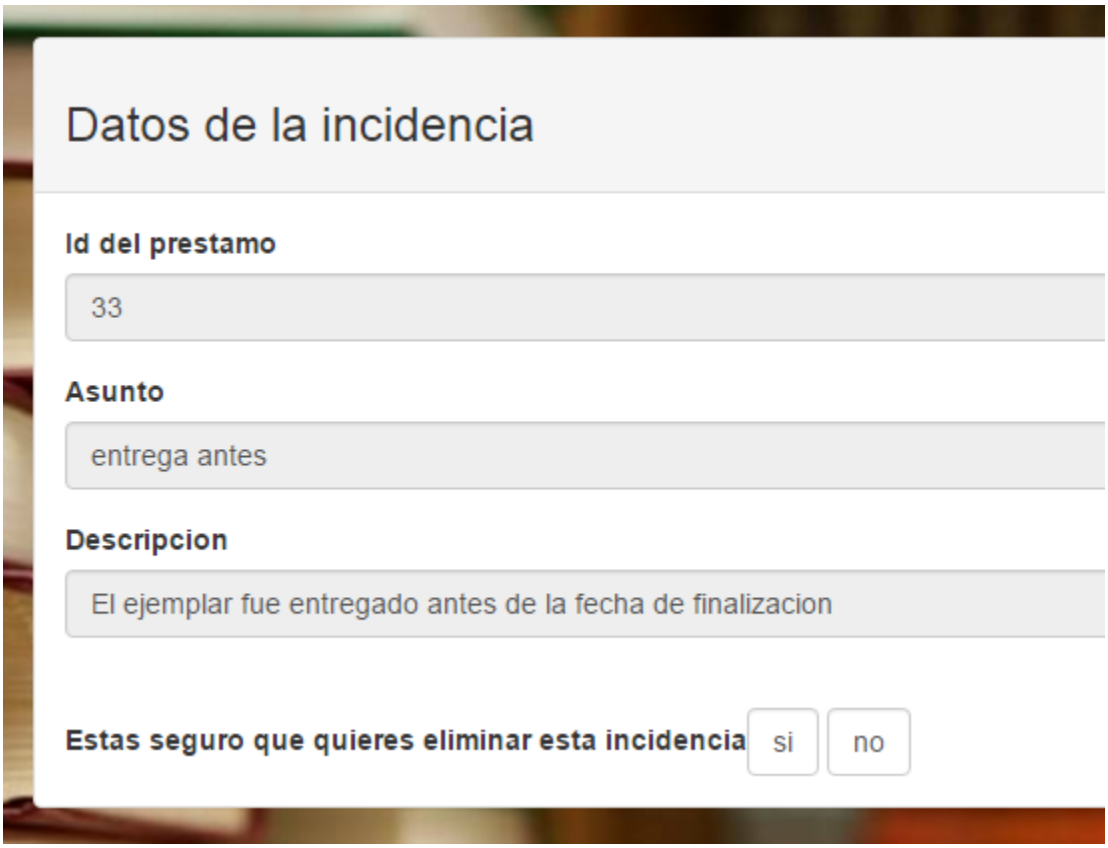
Para abrir la incidencia se le debe indicar el id del préstamo.
Adicionalmente, se puede añadir un asunto y una descripción, no son obligatorios.



6.2 Eliminar incidencia

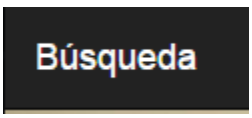
Muestra un listado de los asuntos.

Elegimos el que se quiere eliminar y nos mostrara sus datos:



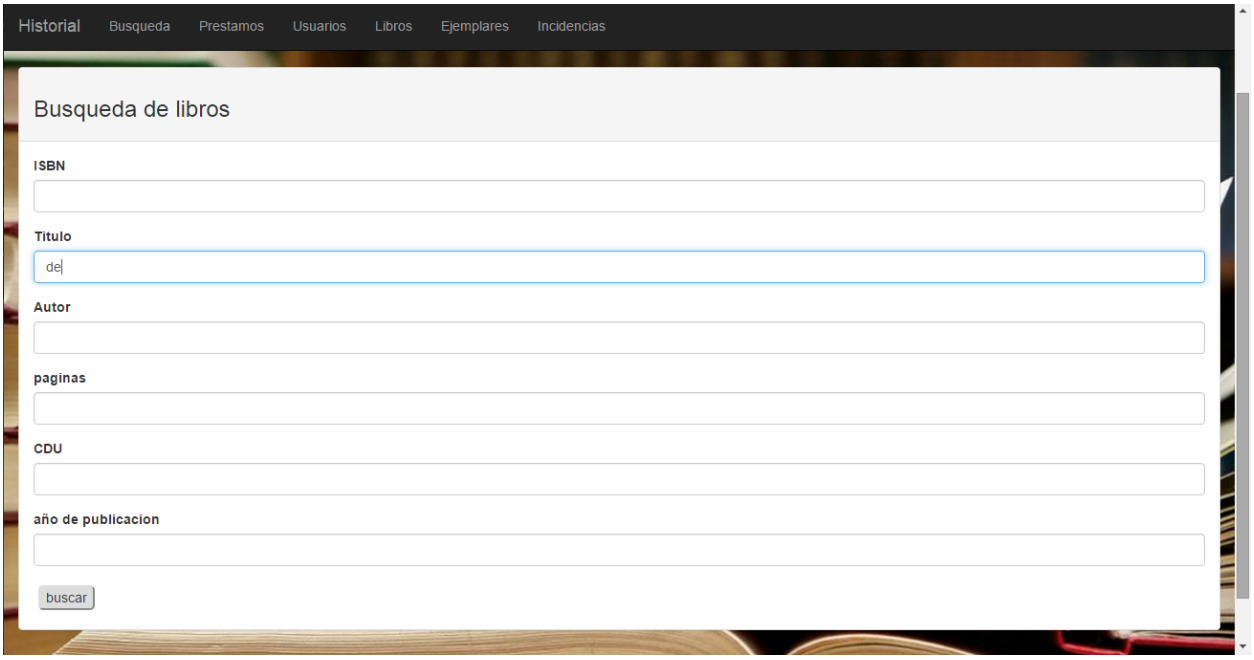
Si elegimos **si** se borrará la incidencia de la aplicación.
Si elegimos **no** nos saldrá la vista inicial.

7. Búsqueda

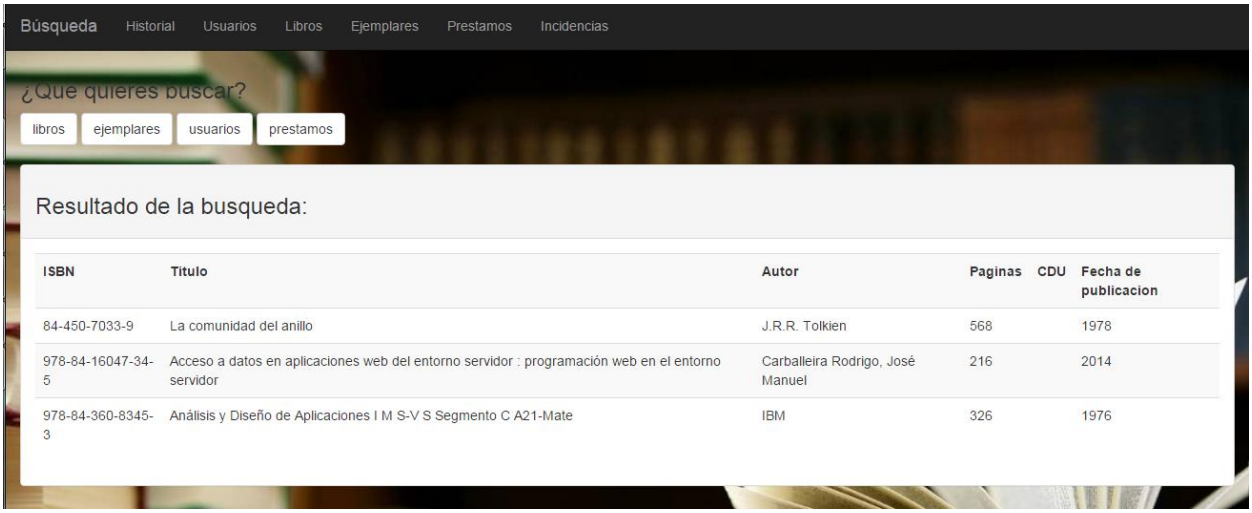


Permite realizar búsquedas de cualquier tipo, ya sea de usuarios, libros, ejemplares o préstamos. Para ello, pulsamos el botón de lo que queremos buscar.

Una vez escogida la opción introducimos los datos que queremos buscar y le damos al botón en la parte inferior.
Como ejemplo de una búsqueda de libros cuyo título contiene “de”.



Pulsamos el botón “buscar” y nos sale este resultado:



De todos los libros:

8. Historial

El historial, muestra con una breve descripción qué acciones se han llevado a cabo por orden de fecha da la opción de limpiarlo, eliminando todas las líneas.