Technical Manual

Jaume Pons Morillas

2n DAW

Library Project

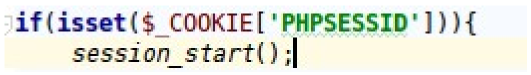
Introduction

This manual will explain how the web application is working. To do so it will explain how the different interactions that the client can do with the server are managed.

Session Management

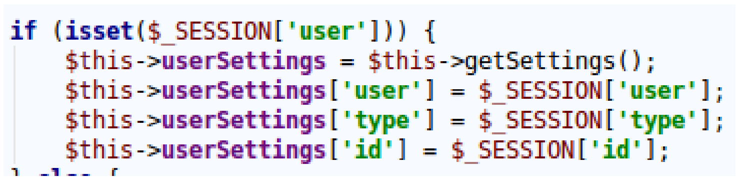
The session is managed from index.php, if a session cookie is reveiced a session will be started.

Then the main contoller will retireve all the information for the session and store it so the controllers can acces it easily.



```
if(isset($_COOKIE['PHPSESSID'])){
    session_start();
```

Index.php

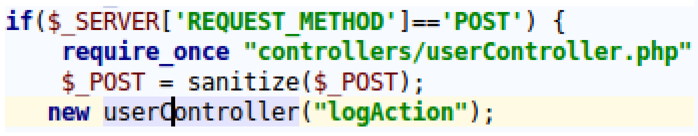


```
if (isset($_SESSION['user'])) {
    $this->userSettings = $this->getSettings();
    $this->userSettings['user'] = $_SESSION['user'];
    $this->userSettings['type'] = $_SESSION['type'];
    $this->userSettings['id'] = $_SESSION['id'];
```

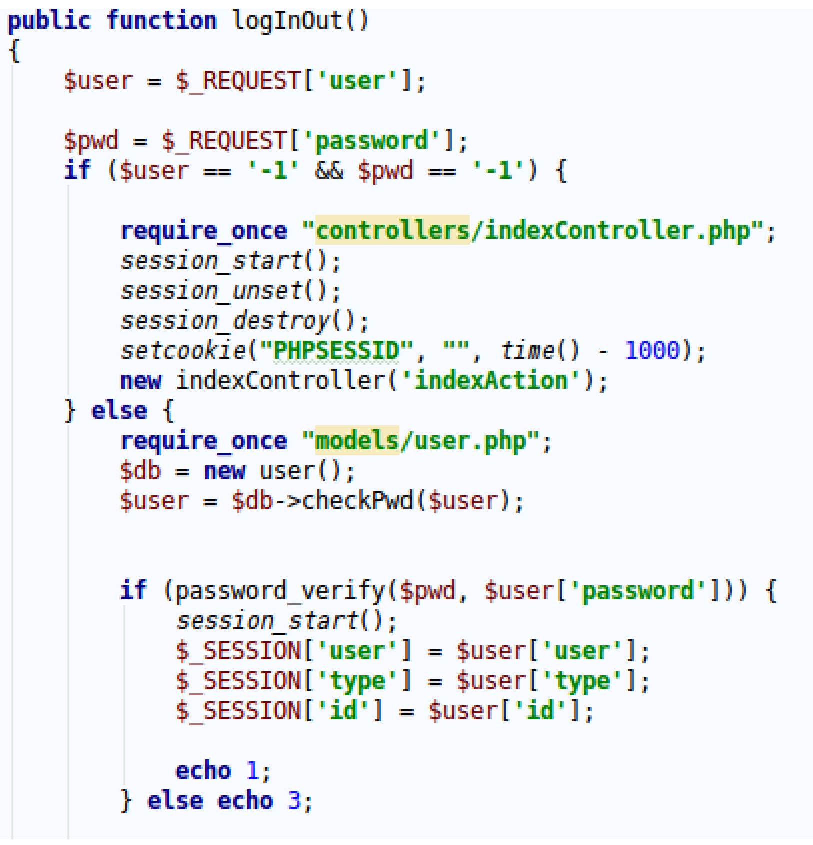
mainController.php

Login

When a user tries to log in the information is sent via ajax to the server. The information arrives to a file called login.php that filters the user input and instanciates the appropiate controller



```
if($_SERVER['REQUEST_METHOD']=='POST') {
    require_once "controllers/userController.php"
    $_POST = sanitize($_POST);
  new userController("logAction");
```



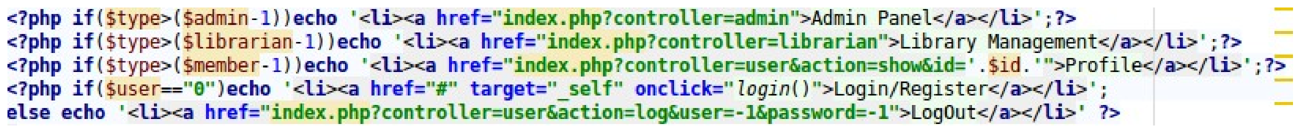
```
public function logInOut()
{
    $user = $_REQUEST['user'];

    $pwd = $_REQUEST['password'];
    if ($user == '-1' && $pwd == '-1') {

        require_once "controllers/indexController.php";
        session_start();
        session_unset();
        session_destroy();
        setcookie("PHPSESSID", "", time() - 1000);
        new indexController('indexAction');
    } else {
        require_once "models/user.php";
        $db = new user();
        $user = $db->checkPwd($user);


        if (password_verify($pwd, $user['password'])) {
            session_start();
            $_SESSION['user'] = $user['user'];
            $_SESSION['type'] = $user['type'];
            $_SESSION['id'] = $user['id'];

            echo 1;
        } else echo 3;
```

Logout

To log out a request to the usercontroller with the logout action is sent, the the logout is handled by the loginOut function above.

Dynamic menu

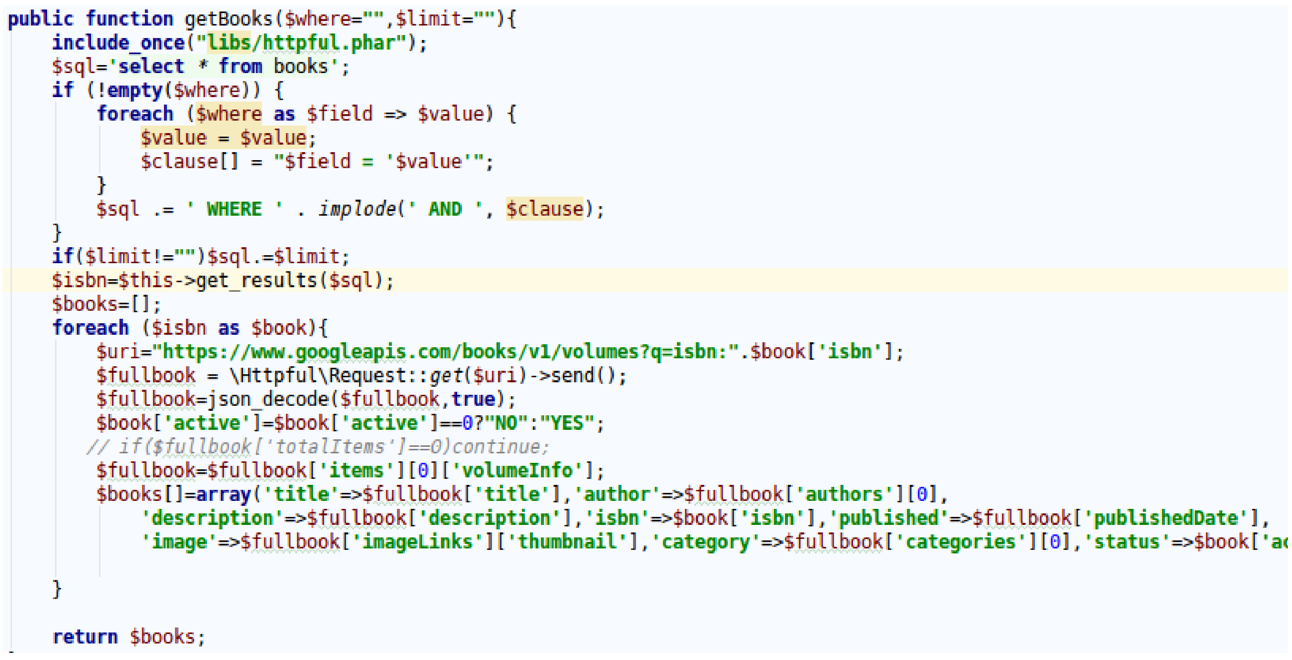
To achieve a dynamic menu the system will check if a user is logged in and retrieve the user type from SESSION. Different options on the menu will be loaded depending on the user type.



```php
<?php if($type>($admin-1))echo '<li><a href="index.php?controller=admin">Admin Panel</a></li>';?>
<?php if($type>($librarian-1))echo '<li><a href="index.php?controller=librarian">Library Management</a></li>';?>
<?php if($type>($member-1))echo '<li><a href="index.php?controller=user&action=show&id='.$id.'">Profile</a></li>';?>
<?php if($user=="0")echo '<li><a href="#" target="_self" onclick="login()">Login/Register</a></li>';
else echo '<li><a href="index.php?controller=user&action=log&user=-1&password=-1">LogOut</a></li>' ?>
```

Display elements

To display the elements the system will get a list of ISBN from the database and then it will realize queries to the googleBooks webservice to retrieve all the information about the book. Then using the boolistWidget class a list of books will be built to display.



```php
public function getBooks($where="",$limit=""){
    include_once("libs/httpful.phar");
    $sql='select * from books';
    if (!empty($where)) {
        foreach ($where as $field => $value) {
            $value = $value;
            $clause[] = "$field = '$value'";
        }
        $sql .= ' WHERE ' . implode(' AND ', $clause);
    }
    if($limit!="")$sql.=$limit;
    $isbn=$this->get_results($sql);
    $books=[];
    foreach ($isbn as $book){
        $uri="https://www.googleapis.com/books/v1/volumes?q=isbn:".$book['isbn'];
        $fullbook = \Httpful\Request::get($uri)->send();
        $fullbook=json_decode($fullbook,true);
        $book['active']=$book['active']==0?"NO":"YES";
       // if($fullbook['totalItems']==0)continue;
        $fullbook=$fullbook['items'][0]['volumeInfo'];
        $books[]=array('title'=>$fullbook['title'],'author'=>$fullbook['authors'][0],
            'description'=>$fullbook['description'],'isbn'=>$book['isbn'],'published'=>$fullbook['publishedDate'],
            'image'=>$fullbook['imageLinks']['thumbnail'],'category'=>$fullbook['categories'][0],'status'=>$book['ac

    }

    return $books;
```

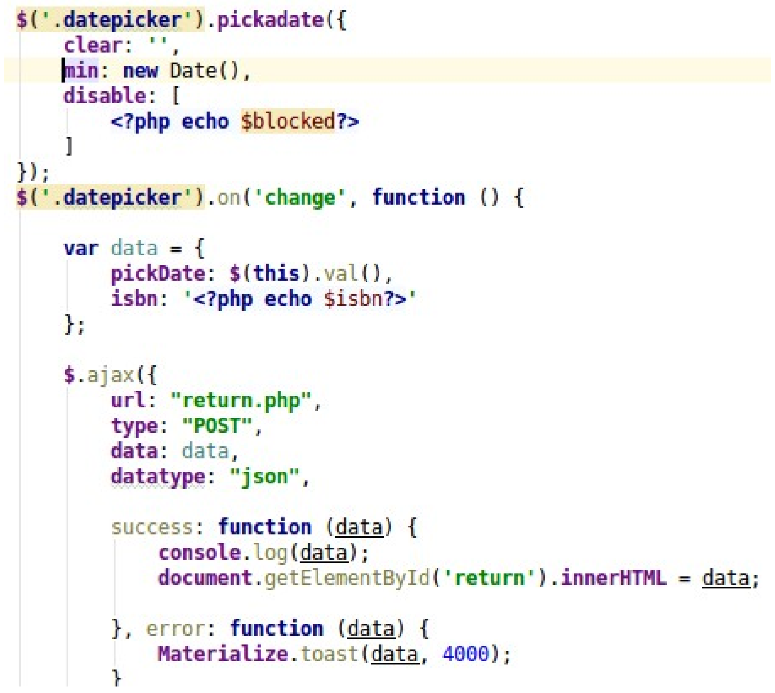
Blocking dates and getting the return date.

To block the dates on the calendar the system performs a query on the database that returns the start and end date of every active booking from the selected book.

By using datepicker.js (a calendar library for JS) the dates are blocked.

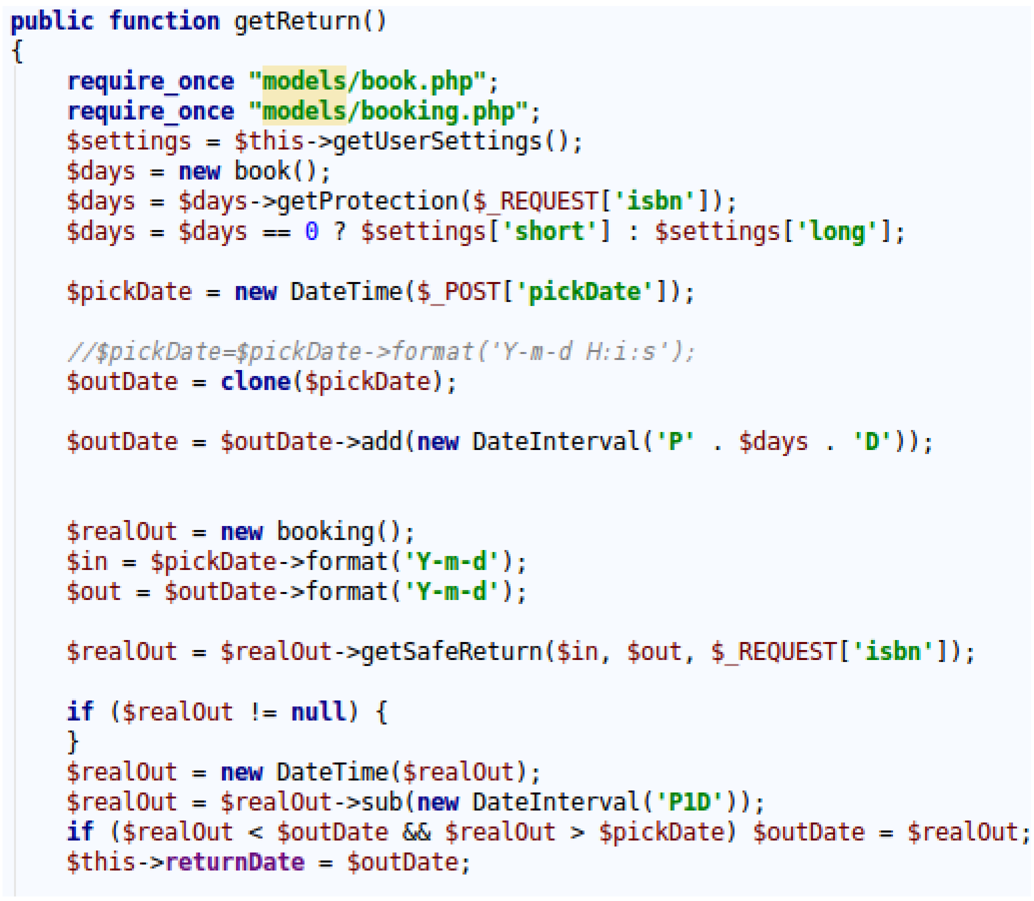
When a user selects a date a request is sent to the server using ajax to get the return date depending on the type of book and if there are other bookings.

When a user confirms a reservation the server checks again that the dates are correct.



```
$('.datepicker').pickadate({
    clear: '',
    min: new Date(),
    disable: [
        <?php echo $blocked?>
    ]
});
$('.datepicker').on('change', function () {

    var data = {
        pickDate: $(this).val(),
        isbn: '<?php echo $isbn?>'
    };

    $.ajax({
        url: "return.php",
        type: "POST",
        data: data,
        datatype: "json",

        success: function (data) {
            console.log(data);
            document.getElementById('return').innerHTML = data;

        }, error: function (data) {
            Materialize.toast(data, 4000);
        }
```

Ajax



```php
public function getReturn()
{
    require_once "models/book.php";
    require_once "models/booking.php";
    $settings = $this->getUserSettings();
    $days = new book();
    $days = $days->getProtection($_REQUEST['isbn']);
    $days = $days == 0 ? $settings['short'] : $settings['long'];

    $pickDate = new DateTime($_POST['pickDate']);

    //$pickDate=$pickDate->format('Y-m-d H:i:s');
    $outDate = clone($pickDate);

    $outDate = $outDate->add(new DateInterval('P' . $days . 'D'));


    $realOut = new booking();
    $in = $pickDate->format('Y-m-d');
    $out = $outDate->format('Y-m-d');

    $realOut = $realOut->getSafeReturn($in, $out, $_REQUEST['isbn']);

    if ($realOut != null) {
    }
    $realOut = new DateTime($realOut);
    $realOut = $realOut->sub(new DateInterval('P1D'));
    if ($realOut < $outDate && $realOut > $pickDate) $outDate = $realOut;
    $this->returnDate = $outDate;
```

Function to calculate the return date.

MVC

The application is using the MVC pattern. All the requests are handled by index.php which filters the user inputs with the sanitize function. Then it instanciates the aproppiate controller. The controller instanciates the model to retrieve the information and then instanciates the view class with the requiered information. The view class will extract the data which is arriving as an associative array and will build the view which is sent to the user.

There are 3 superclases: db.php, mainController.php and view.php.

The database superclass has all the information to acces the database, including methods to retrieve, insert, update or delete data. It has also a function to clean the data to insert on a database.

The mainController from which every controller has to inherit has one important method: getSettings(). This method retrieves the library settings from a json file. On the constructor the information stored in session toghether with the information from the settings file will be stored on the userSettings atribute.

The view class from which every view inherits has only the template atribute with its getters and setters.

Subclasses and their functions:

Controllers:

- Admin controller: Manages the exclusive admin actions, viewing the admin panel and changing the parameters.
- BookController: manages all the actions related to books, displaying books, adding books, updating books and showing the histroy of a book.
- BookingController: This class manages all the actions related to bookings, displaying the booking page, calculating the return date, save a booking, getting all the bookings due to finish on a day and returning a book.
- CatalogController: Manages all the actions related to the catalogue: viewing the cataloge and searching for a book.
- ErrorController: this controller is used to display custom error messages.
- IndexController: maganages the homepage view and the register page.
- LibrarianController: used to show the librarian page.
- UserController: manages all the action related to users: adding a new user, showing a users profile, showing the edit profile page, updating the user profile, log in and log out and showing all the bookings of a user.

Models:

- book.php: class to interact with the books table.
- Booking.php: class to interact with the bookings table
- user.php: class to interact with the users table.

Views:

There's a view class for every page that is displayed to the user, their names are selfexplanatory.

Widgets:

- bookListWidget: this widget is used to display the list of books on the librarian page.
- BookWidget: used to display the books on the catalogue.
- HistWidget: used to display the history of bookings from a user and from a book.
- UserWidget: used to build a list of users for the librarian page.

AJAX

There are some actions performed by AJAX on the application: Log in, adding a book, and returning a book.

The ajax functions call a intermediary file on the server (acts as a index.php) that filters the input and then instanciates the apropiate class. It also prepares and sends the results back to the client.

Things to improve

There’s is a system implemented to check if a user has permission to perform an action, but it’s not implemented on every action.

Another big improvement would be the use of an htacces to make urls cleaner.

Even if the google books webservice is really good is too slow to use. A migration to the goodreads webservice which is faster would improve the speed of the application.

On the librarian page every tab should make an AJAX request to get the information to display, instead of loading everything from the start.

Changing the database class to use prepared statements would improve the security of the system, and make it easier to implement new functionalities.