SERVER-SIDE WEB PROGRAMMING UNIT5: STORING INFORMATION WITH DATABASES

CFGS - DAW 2

Index

- Completing books functionality:
  - Delete
  - View
  - Add

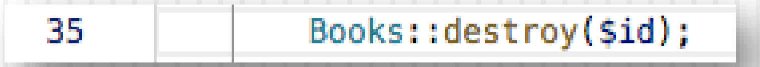
1. Delete a book

- Until now we have just make a full query to the DB and show list of books to the user.
- Let´s include the other 3 basic operations: *delete, view and add.*

1. For the delete action, let´s define the corresponding *delete* method inside of the controller.
2. Inside of the method: 1st take the id that is passed through the URL, reading the arguments of the method (Hint: *func_get_args()*)

1. Delete a book

3. There is an eloquent method called *destroy, used* in order to delete items from the DB. Use it this way:



```
Books::destroy($id);
```

4. After that, what we need is to redirect the application again to the home page. In order the new query to be built enterely let´s use *header()* php method to redirect the application to our ROOT_URL.

1. Delete a book

5. Be aware, that in this case, we do not need to create the corresponding view. However, what we need is to create the corresponding link inside of the *index.html.php* view calling to:

   EX.: …books/delete/5

Test your delete action link and check that it deletes the entity in the DB and that you redirect the web page to the home.

2. View a book

- Let´s follow the same pattern in order to create the view action:

1. For the view action, let´s define the corresponding *read* method inside of the controller. We will not used *view*, because it would understand you want to overwrite it from the parent.
2. Inside of the method: 1st take the id that is passed through the URL, reading the arguments of the method (Hint: *func_get_args()*)

2. View a book

3. Find in the Eloquent´s documentation, ***which is the method*** in order to find an **Entity by its id**, and use it. Remember, now you will have to save the returning value inside of a variable, that will be passed to the corresponding view.
4. Finally, inside of the view method, load the view.html.php view, and create it inside of the corresponding views folder.

2. View a book

5. The only thing we are missing, is changing the *index.html.php* file, in order to add a link to the corresponding action for each of the entries. You can add it in the titles:

   EX.: …books/read/5

Test your view action link and check that it redirects to the corresponding view.

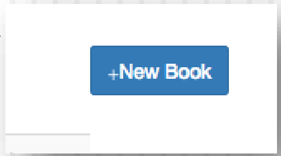
3. Add a book

- Before continuing, we need to have clear that for the same …books/add action, we can have two types of queries:

1. A GET query, that will mean that the server has to present the corresponding form.
2. A POST query, that will mean that the server has to:
    1. Check the incoming data
    2. Insert Data
    3. Redirect the user to the home page.

3. Add a book

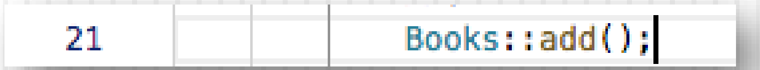
1. Create the corresponding method in the controller called *add*.
2. Inside of the method write a condition that checks in the REQUEST_METHOD is a GET or not.
3. In case it is, send the user to an *add.html.php* view.
4. That view, will include a form with the corresponding fields. Create it inside of the corresponding folder.
5. Also, add a button that gets you into the form in your *index.html.php* view:



Before controlling the POST query, check that the books/add action with a GET, sends the user to the form you have created.

3. Add a book

6. Inside of the condition of the add method, for the POST case:
   1. The first thing is going to be, to call to a new method we will create afterwards that will belong to our Books model: add. Note: be aware that we are calling it statically…

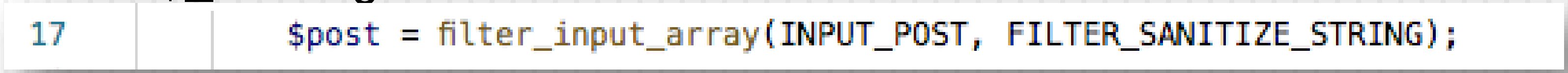


```
Books::add();
```

   2. After that, redirect the user to the home page.

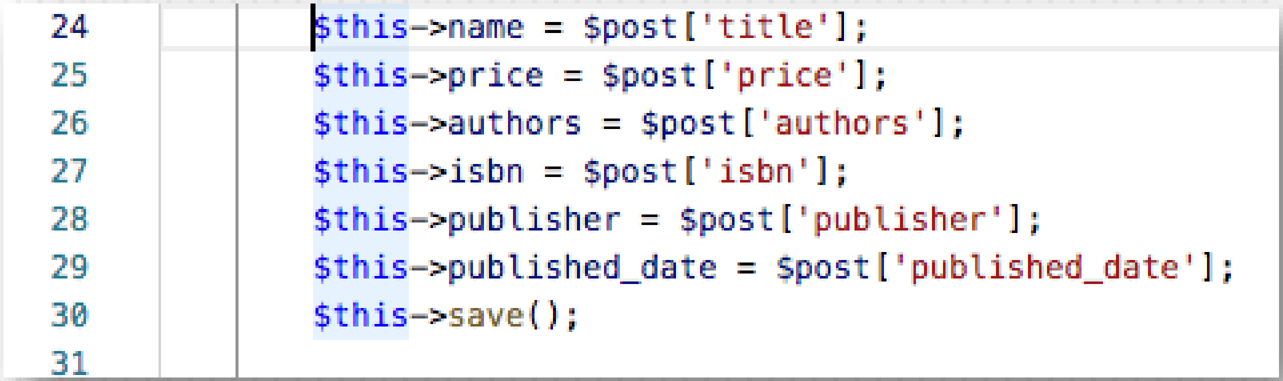
3. Add a book

7. Let´s modify our Books model in order it to include *add* action for our model:
   1. The first thing is going to be, to be able to read data from the $_POST global variable and sanitize it.



```
        $post = filter_input_array(INPUT_POST, FILTER_SANITIZE_STRING);
```

   2. Let´s add a condition that cheks if all the fields are filled and in that case:



```
        $this->name = $post['title'];
        $this->price = $post['price'];
        $this->authors = $post['authors'];
        $this->isbn = $post['isbn'];
        $this->publisher = $post['publisher'];
        $this->published_date = $post['published_date'];
        $this->save();

```

3. Add a book

Fill in a form and check that the information is inserted into the DB and that the user is redirected to the home page.

Note: probably you will be getting an error saying that you are calling to a method that is not static statically from our controller. Check this. Add the "scope" keyword before the name of the method, as in the example.