Jesuitas
Logroño

SERVER-SIDE WEB PROGRAMMING UNIT3: STORING INFORMATION WITH DATABASES

CFGS - DAW 2

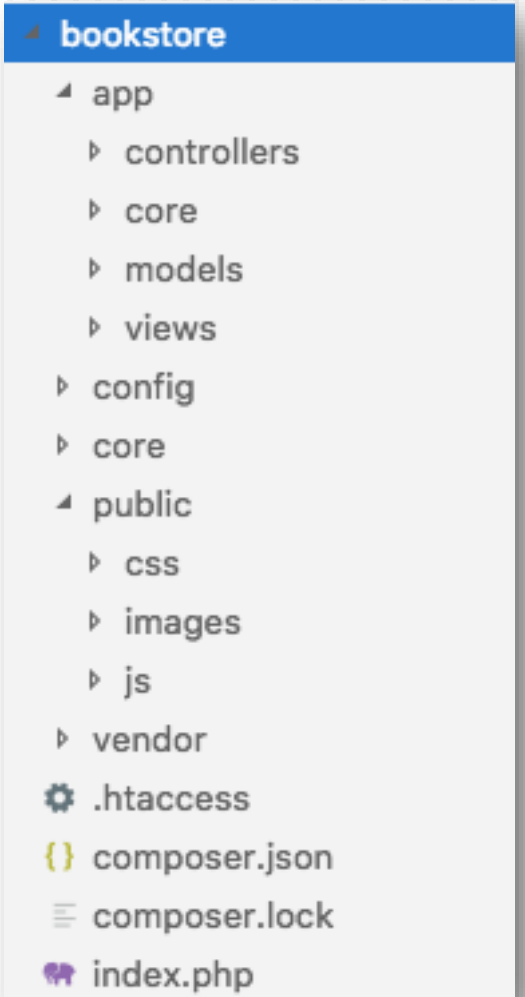
Index

- Project structure
- Routing
- Books controller
- Model – ORM
- Views

1. Project Structure

- We are going to start building our application. In order to do so, follow these steps:

1. Create this project structure:



1. Project Structure

2. Conventions we are going to follow:
   - mySQL tables will always be lowercase and plural e.g. items, cars, products, users
   - Models will usually start with capital letters e.g. Products, Cars (depending on the framework you will find them in plural or not)
   - Controllers will always have “Controller” appended to them. e.g. ProductsController, CarsController

1. Project Structure

3. Create a new DB called ‘bookstore’ and a table inside called ‘books’ like this:

#	Nombre	Tipo	Cotejamiento
1	**id**	int(11)	
2	**name**	varchar(50)	utf8_general_ci
3	**price**	float	
4	**authors**	varchar(20)	utf8_general_ci
5	**isbn**	varchar(20)	utf8_general_ci
6	**publisher**	varchar(20)	utf8_general_ci
7	**published_date**	smallint(6)	
8	**updated_at**	timestamp	
9	**created_at**	timestamp	

2. Routing

1. Copy this inside your .htaccess file:

```
1   Options -Indexes
2   Options -MultiViews
3   RewriteEngine On
4
5   RewriteCond %{REQUEST_FILENAME} !-d
6   RewriteCond %{REQUEST_FILENAME} !-f
7
8   RewriteRule ^(.+)$ index.php?url=$1 [QSA,L]
```

- Line1: Security reasons, no listing of folders or subfolders
- Lines 5 and 6: make sure that the **path** requested is not a filename or directory.
- Line 8: redirects all paths. Example:

  .../public/**books/addbook** → .../index.php?url=**books/addbook**

2. Routing

2. This is going to be the code for our index.php:

```
1   <?php
2   // Start Session
3   //session_start();
4
5   // Composer autoloader
6   require_once 'vendor/autoload.php';
7
8   require_once 'config/config.php';
9   require_once 'config/database.php';
10
11
12  $app = new App;
```

- Leave commented lines 3 and 9 (for now).

2. Routing

3. *config.php* file inside of config folder:

```php
<?php

// Define DB Params
define("DB_HOST", "localhost");
define("DB_USER", "root");
define("DB_PASS", "root");
define("DB_NAME", "bookstore");

// Define URL
define("ROOT_PATH", "/test/bookstore/");
define("ROOT_URL", "http://dwes.local/test/bookstore/");

```

2. Routing

4. *app/core/App.php* class:

```php
<?php
class App
{
    protected $controller = 'Books';
    protected $method = 'index';
    protected $params = [];

    public function __construct()
    {
        $url = $this->parseUrl();
        $url[0] = isset($url[0]) ? ucfirst($url[0]) : 'Books';
        if (file_exists('app/controllers/' . $url[0] . 'Controller.php')) {
            $name = $url[0];
            $this->controller = $url[0] . 'Controller';
            $this->controller = new $this->controller($name);
            unset($url[0]); //remove from the array
        }
        if (isset($url[1])) {
            if (method_exists($this->controller, $url[1])) {
                $this->method = $url[1];
                unset($url[1]);
            }
        }
        $this->params = $url ? array_values($url) : [];
        if (isset($_POST) && !empty($_POST)) {
            $this->params = $_POST;
        }
        call_user_func_array([$this->controller, $this->method], $this->params);
    }

    public function parseUrl()
    {
        if (isset($_GET['url'])) {
            $url = explode('/', filter_var(rtrim($_GET['url'], '/'), FILTER_SANITIZE_URL));

            return $url;
        }
    }
}
```

2. Routing

Test the routing this way:

1. Create a new BooksController class inside controller folder.
2. Inside of the class create an index method, where just echo the message you want, so that you are aware you are able to call it, using the FW!!

3. Books controller

1. Inside app/core folder we are going to create a parent Controller class:

```php
<?php

class Controller
{
    protected $controller;

    public function __construct($name)
    {
        $this->controller = $name;
    }

    public function view($view, $data = [])
    {
        require_once 'app/views/' . $this->controller . '/' . $view . '.php';
    }
}
```

3. Books controller

2. Modify BooksController in order it to extend from the recently created Controller class.
3. Inside you have to declare 3 functions:
   - index
   - add
   - delete

Test that you are able to call to the 3 methods changing the URL and call to the corresponding views (right now the content of the view is not important).
For instance: .../books/index (URL) → create the corresponding index.html.php file and call it from the index method from the controller.

```
6    public function index()
7    {
8        $this->view('index.html', ['books' => "Mis libros"]);
9    }
```

4. Model - ORM

- In order to continue, we need to create the model, in order to be able to work with the DB.
- Actually we are going to use an ORM, called Eloquent (used by Laravel, for instance).
- ORM → is a programming technique for converting data between incompatible type systems using object-oriented programming languages (from OO to Relational model):
  - Easy to use - Abstraction
  - Security
  - Reduce the amount of code

4. Model - ORM

- As you already know, in order to use in PHP third parties libraries you will need to use composer.
  - Eloquent is a third-party library: illuminate/database
  - We need to make a reference to that library in our project, in order to manage that dependency… in order to do so, we are going to use composer!

4. Model - ORM

1. If we open a new command line and introduce this command, you will add a dependecy inside the composer.json file using the command line:

```
composer require illuminate/database
```

2. You can see how:
   - It also installs the corresponding autoloading files and dependencies/libraries inside of vendor folder, in case they are not.
   - *composer.json* file has changed.
   - Also, when you launch the command, it is going to download not only the required library, but also its own dependencies.

4. Model - ORM

3. Next thing we have to do, is to load the new dependencies that we have defined inside the composer. In order to do so, we are going to use the autoload file included by composer (*index.php* file):

```
5   // Composer autoloader
6   require_once 'vendor/autoload.php';
7
```

4. Model - ORM

4. Another very common task to any FW, is to separate the data related to connections using configurations files. Follow these steps:
   - Inside *config* folder, create a *database.php* file. Leave it empty for the moment.
   - Do not forget to require this file inside the *index.php file*:

```
9    require_once 'config/database.php';
```

4. Model - ORM

5. Another thing, we are going to make is to use the autoload capability of the composer file, so that our model/controller/core classes are automatically loaded when autoload.php file from composer is called:

```
1  {
2      "require": {
3          "illuminate/database": "^5.4",
4          "setasign/fpdf": "^1.8"
5      },
6      "autoload": {
7          "classmap": [
8              "app/core", "app/models", "app/controllers"
9          ]
10
11     }
12 }
```

- This will make that all model classes that we use in our application to be globally available.
- We will learn how to access them afterwards.

- After adding lines from 6 until 11, go into the command line and launch this:

```
composer dump-autoload
```

4. Model - ORM

6. Inside *database.php* file we are going to set up the configuration needed in order to use the ORM in our project:

```php
<?php

use Illuminate\Database\Capsule\Manager as Capsule;

$capsule = new Capsule();

$capsule->addConnection([
    'driver' => 'mysql',
    'host' => DB_HOST,
    'username' => DB_USER,
    'password' => DB_PASS,
    'database' => DB_NAME,
    'charset' => 'utf8',
    'collation' => 'utf8_general_ci',
    'prefix' => '',
]);

$capsule->bootEloquent();
```

- For the moment, we have just set up Eloquent in order to make use of it, but we have not yet!

4. Model - ORM

7. Now, create a new model called Books.php model file.
   - Once, you have done that, just extend Books class from Eloquent class like this:

```
<?php

use Illuminate\Database\Eloquent\Model as Eloquent;

class Books extends Eloquent
{



}
```

4. Model - ORM

8. Our Books model is already able to use *Eloquent* class method:
   - The first thing we are going to do is to change the previous ***index*** method from our **BooksController** functionality but calling to the corresponding Eloquent method:

```
6    public function index()
7    {
8        $books = Books::all();
9        $this->view('index.html', ['books' => $books]);
10   }
```

   - In this case, you are making use of *all()* method from Eloquent.
   - We are using directly the static methods from Books… **This is because of the AUTOLOAD process we made earlier!!!**

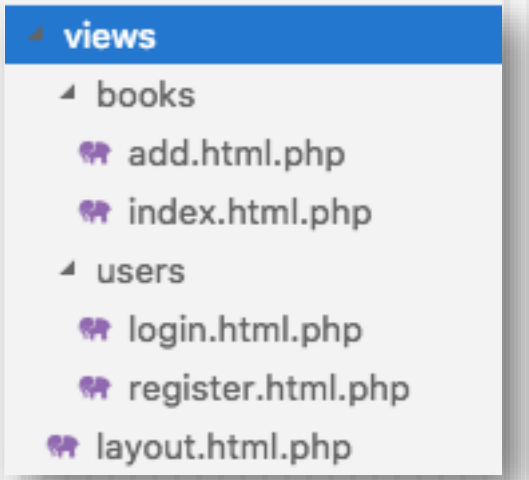
4. Model - ORM

In order to test all this…:

1) Insert some data in the DB.
2) Comment the second line from the index method and make a print_r of the $books… to see if you are getting something from the DB

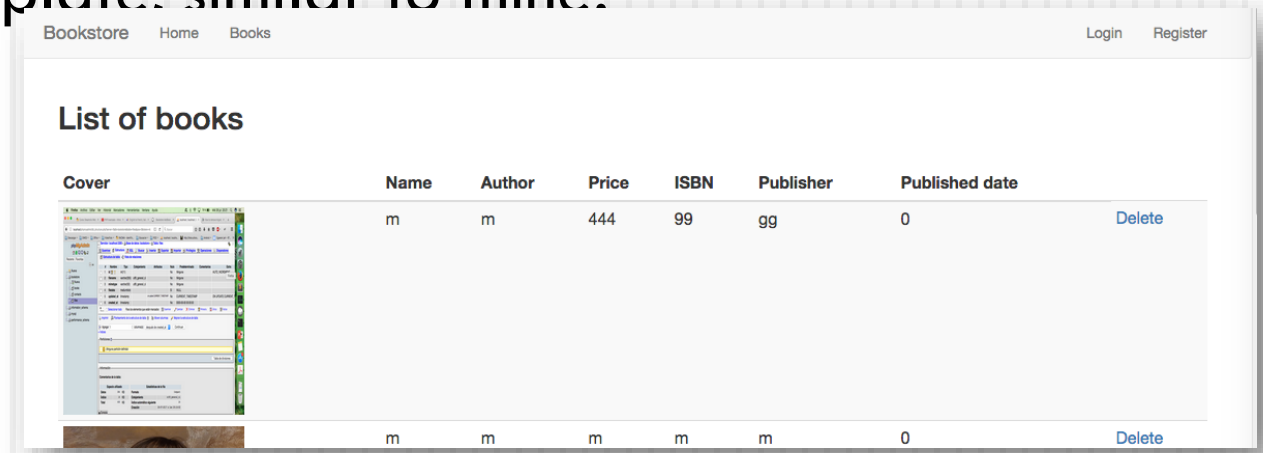
5. Views

- In order to create the corresponding view for our books/index action, we need to setup the general layout (the same for all views) and the corresponding view.



5. Views

1. For the **general layout** take an easy bootstrap template, similar to mine:



1. Copy the corresponding jquery, css, fonts and whatever into your css/js/fonts folders.
2. Change the references to the proper place in your project.
3. Do not forget to add this line in order to fill it with the content:

```
40    <div class="container">
41
42    <div class="row">
43        <?=$content?>
44    </div>
```

5. Views

4. Use a pretty bootstrap table to print the information of every book inside of *index.html.php* file.
   - This time (and for every view) you cannot forget these lines at the beginning and and the end of the script:

```
1  <?php ob_start();?>
2      <div style="width: 95%; margin: 0 auto;">
3        <h2>List of books</h2>
4         <?php if (isset($_SESSION['is_logged_in
5            <a class="btn btn-sm btn-primary pul
6        <?php endif;?>
7        <div class="table-responsive">
```

```
38     </div>
39     <?php $content = ob_get_clean()?>
40     <?php include 'app/views/layout.html.php'?>
41
```

4. Model - ORM

Can you test the books/index action (or the home) and the list of books is printed properly with the layout you have chosen?