

Lab # 09: Laravel MVC Framework

OBJECTIVES OF THE LAB

This lab aims at the understanding of:

- *PHP MVC Framework in Laravel*
 - *Implementation of simple CRUD in Laravel*
-

PHP MVC FRAMEWORK IN LARAVEL

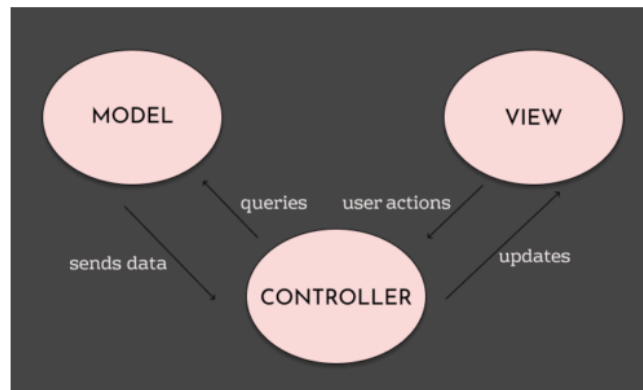


Figure 9.1 – Model-View-Controller Framework

Figure 9.1 depicts the MVC Framework where M stands for Model, V stands for View and C stands for Controller. A **Model** is a representation of a real-life instance or object in our code base. It can either be a database or JSON File or some other source. The **View** represents the user interface through which the user interacts with our application. It contains HTML or the presentation markup. It can also have logic e.g. loops and conditionals. Template engines are used to embed logic in views. Laravel has Blade template engine that is used for adding logic inside the views. When a user takes an action, the **Controller** handles the action and updates the model if necessary.

Note 1: By default, Laravel supports PDO PHP Extension.

Note 2: Currently, Laravel supports four different DBMS including MySQL, SQLite, Postgre SQL, and SQL Server.

A SIMPLE CRUD IN LARAVEL

This example uses MySQL DBMS and PDO PHP Extension.

Step 1: Installing the Laravel 5.2 Framework.

```
C:\wamp\www\lv11>composer create-project --prefer-dist laravel/laravel="5.2.31" SampleLv1
Installing laravel/laravel (v5.2.31)
- Installing laravel/laravel (v5.2.31): Loading from cache
Created project in SampleLv12
> php -r "copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies (including require-dev)
```

Step 2: Create the database.

```
mysql> create database SampleLv1;
Query OK, 1 row affected (0.04 sec)
```

Step 3: Setup a MySQL database in .env file.

- Open file: C:\wamp\www\lv11\SampleLv1\.env and do following changes:

```
6 DB_CONNECTION=mysql
7 DB_HOST=127.0.0.1
8 DB_PORT=3306
9 DB_DATABASE=SampleLv1
10 DB_USERNAME=root
11 DB_PASSWORD=
```

- Save the file.

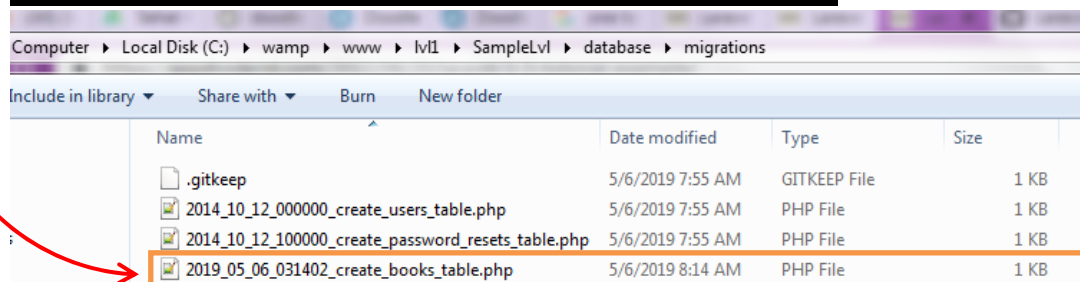
Step 4: Migrate the build-in tables in our database.

```
C:\wamp\www\lv11\SampleLv1>php artisan migrate
Migration table created successfully.
Migrated: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_100000_create_password_resets_table

mysql> use SampleLv1;
Database changed
mysql> show tables;
+-----+
| Tables_in_samplelv1 |
+-----+
| migrations           |
| password_resets      |
| users                |
+-----+
3 rows in set (0.00 sec)
```

Step 5: Create a **MODEL** as well as migration file for our Books table.

```
C:\wamp\www\lv11\SampleLv1>php artisan make:model Books -m
Model created successfully.
Created Migration: 2019_05_06_031402_create_books_table
```



Name	Date modified	Type	Size
.gitkeep	5/6/2019 7:55 AM	GITKEEP File	1 KB
2014_10_12_000000_create_users_table.php	5/6/2019 7:55 AM	PHP File	1 KB
2014_10_12_100000_create_password_resets_table.php	5/6/2019 7:55 AM	PHP File	1 KB
2019_05_06_031402_create_books_table.php	5/6/2019 8:14 AM	PHP File	1 KB

- Open the **2019_05_06_031402_create_books_table.php** file and update it as follows:

```

public function up()
{
    Schema::create('books', function (Blueprint $table) {
        $table->increments('id');
        $table->string('name');
        $table->string('author');
        $table->integer('price');
        $table->timestamps();
    });
}

```

- Save the file. Now, our books table contains six attribute: id, name, author, price, created_at, and updated_at. Let's migrate it and view in database.

```

C:\wamp\www\lv11\SampleLv1>php artisan migrate
Migrated: 2019_05_06_031402_create_books_table

```

```

mysql> show tables;
+-----+
| Tables_in_samplelv1 |
+-----+
| books                |
| migrations           |
| password_resets      |
| users                |
+-----+
4 rows in set (0.00 sec)

```

```
mysql> describe books;
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_increment
name	varchar(255)	NO		NULL	
author	varchar(255)	NO		NULL	
price	int(11)	NO		NULL	
created_at	timestamp	YES		NULL	
updated_at	timestamp	YES		NULL	

```
6 rows in set (0.01 sec)
```

Step 6: Make one **VIEW** file to add the form data in the database.

- Go to C:\wamp\www\lv11\SampleLv1\resources\views. Create folder for the books model. Name it books. Go inside the newly created folder.
- Create the file: **create.blade.php**. Add the following code in it:

```

<!-- create.blade.php -->

1. <!DOCTYPE html>
2. <html>
3. <head>
4.     <meta charset="utf-8">
5.     <title>Laravel 5.5 CRUD Tutorial</title>
6.     <link rel="stylesheet" href="{{asset('css/app.css')}}">
7. </head>
8. <body>
9.     <div class="container">
10.    <h2>Add a Book</h2><br />
11.    <form method="post" action="">
12.        <div class="row">
13.            <div class="col-md-4"></div>

```

```

14.         <div class="form-group col-md-4">
15.             <label for="name">Name:</label>
16.             <input type="text" class="form-control" name="name">
17.         </div>
18.     </div>
19.     <div class="row">
20.         <div class="col-md-4"></div>
21.         <div class="form-group col-md-4">
22.             <label for="name">Author:</label>
23.             <input type="text" class="form-control" name="author">
24.         </div>
25.     </div>
26.     <div class="row">
27.         <div class="col-md-4"></div>
28.         <div class="form-group col-md-4">
29.             <label for="price">Price:</label>
30.             <input type="text" class="form-control" name="price">
31.         </div>
32.     </div>
33. </div>
34.     <div class="row">
35.         <div class="col-md-4"></div>
36.         <div class="form-group col-md-4">
37.             <button type="submit" class="btn btn-success" style="margin-left:38px">Add
Book</button>
38.         </div>
39.     </div>
40. </form>
41. </div>
42. </body>
43. </html>

```

- Go to following folder: C:\wamp\www\lv1\SampleLv1\public. Create the folder css in it and store the given app.css file in it.

Step 7: Create one **CONTROLLER and route to display the Books form.**

- `C:\wamp\www\lv1\SampleLv1>php artisan make:controller BookController --resource`
Controller created successfully.
- It will generate one controller file called BookController.php in C:\wamp\www\lv1\SampleLv1\app\Http\Controllers.
- Next, in the routes configuration file — C:\wamp\www\lv1\SampleLv1\app\Http\routes.php— add the following to define a Books resource route:

```
Route::resource('books', 'BookController');
```

- Save the file. That single route definition will define all of the routes related to our Books resource. It can be viewed as follows:

```
c:\wamp\www\lv1\SampleLv1>php artisan route:list
```

Domain	Method	URI	Name	Action	Middle
	GET HEAD	/		Closure	web
	POST	books	books.store	App\Http\Controllers\BookController@store	web
	GET HEAD	books	books.index	App\Http\Controllers\BookController@index	web
	GET HEAD	books/create	books.create	App\Http\Controllers\BookController@create	web
	DELETE	books/{books}	books.destroy	App\Http\Controllers\BookController@destroy	web
	PUT PATCH	books/{books}	books.update	App\Http\Controllers\BookController@update	web
	GET HEAD	books/{books}	books.show	App\Http\Controllers\BookController@show	web
	GET HEAD	books/{books}/edit	books.edit	App\Http\Controllers\BookController@edit	web

- Next go to BookController.php file and add into create() function some code.

```
public function create()
{
    // updated
    return view('books.create');
}
```

- Next, start Laravel Development server: **php artisan serve.**
- Go to following: <http://localhost:8000/books/create>.

Add a Book

Name:

Author:

Price:

Step 8: Put the Laravel 5.5 Validation in Product Form.

- Open the C:\wamp\www\lv1\SampleLv1\resources\views\books\create.blade.php file. In file, remove the line No. 11 and add the following code after line No. 10.

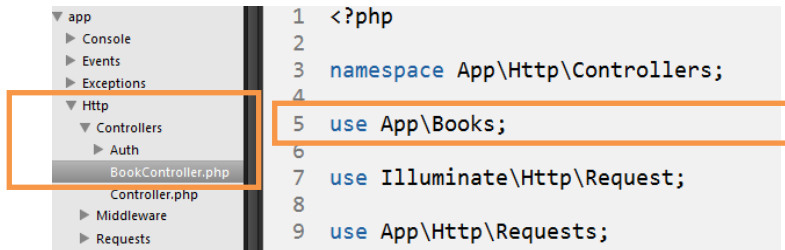
```
11. @if ($errors->any())
12. <div class="alert alert-danger">
13.     <ul>
14.         @foreach ($errors->all() as $error)
15.             <li>{{ $error }}</li>
16.         @endforeach
17.     </ul>
18. </div><br />
19. @endif
20. @if (\Session::has('success'))
21. <div class="alert alert-success">
22.     <p>{{ \Session::get('success') }}</p>
23. </div><br />
24. @endif
25. <form method="post" action="{{url('books')}}">
```

26. {{csrf_field()}}

- Next, go to C:\wamp\www\lv1\SampleLv1\app\Books.php. Add the following code:

```
class Books extends Model
{
    // update
    protected $fillable = ['name', 'author', 'price'];
}
```

- Save the file.
- Type the following line at the starting of BookController.php file: use App\Books;



- Next, update the store function in it as follows:

```
40 public function store(Request $request)
41 {
42     // updated
43     $books = new Books;
44     $books->name = $request->get('name');
45     $books->author = $request->get('author');
46     $books->price = $request->get('price');
47     $books->save();
48
49     return back()->with('success', 'Book has been added');
50 }
```

- Now check the working. Open the page: <http://localhost:8000/books/create>. Provide book name, author name, and price. Press Add Book. Once the book is successfully added, then it'll be shown.

Add a Book

Book has been added

Name:

Author:

Price:

Add Book

- Check in database.

```
mysql> select * from books;
Empty set (0.00 sec)
```

```
mysql> select * from books;
```

id	name	author	price	created_at	updated_at
1	Modern Database Management	McFadden	200	2019-05-06 04:20:03	2019-05-06 04:20:03

1 row in set (0.00 sec)

Step 9: Make an index page to list the books

```
18     public function index()  
19     {  
20         // updated  
21         $books = Books::all()->toArray();  
22         return view('books.index', compact('books'));  
23     }
```

- In C:\wamp\www\lv1\SampleLv1\resources\views\books, create one blade file called index.blade.php file and put the following code in it.

```
1. <!-- index.blade.php -->  
2.  
3. <!DOCTYPE html>  
4. <html>  
5.     <head>  
6.         <meta charset="utf-8">  
7.         <title>Index Page</title>  
8.         <link rel="stylesheet" href="{{asset('css/app.css')}}">  
9.     </head>  
10.    <body>  
11.        <div class="container">  
12.            <br />  
13.            @if (\Session::has('success'))  
14.                <div class="alert alert-success">  
15.                    <p>{{ \Session::get('success') }}</p>  
16.                </div><br />  
17.            @endif  
18.            <table class="table table-striped">  
19.                <thead>  
20.                    <tr>  
21.                        <th>ID</th>  
22.                        <th>Name</th>  
23.                        <th>Author</th>  
24.                        <th>Price</th>  
25.                        <th colspan="2">Action</th>  
26.                    </tr>  
27.                </thead>  
28.                <tbody>  
29.                    @foreach($books as $book)  
30.                        <tr>  
31.                            <td>{{ $book['id'] }}</td>  
32.                            <td>{{ $book['name'] }}</td>  
33.                            <td>{{ $book['author'] }}</td>  
34.                            <td>{{ $book['price'] }}</td>
```

```

35.     <td><a href="{{action('BookController@edit', $book['id'])}}" class="btn btn-
warning">Edit</a></td>
36.     <td>
37.         <form action="{{action('BookController@destroy', $book['id'])}}" method="post">
38.             {{csrf_field()}}
39.             <input name="_method" type="hidden" value="DELETE">
40.             <button class="btn btn-danger" type="submit">Delete</button>
41.         </form>
42.     </td>
43. </tr>
44. @endforeach
45. </tbody>
46. </table>
47. </div>
48. </body>
49. </html>

```

- Now go to following: <http://localhost:8000/books>. Books are listed.

ID	Name	Price	Action
1	Modern Database Management	McFadden	200
			<div>Edit</div> <div>Delete</div>

Step 10: Delete the books

- Both Edit and Delete in above example works nothing as of now.
- In this step, delete code is written so that the respective book be deleted. Note: only change is needed in the Controller code. No need to write any view for this command. Add the following code in BookController.php.

```

94     public function destroy($id)
95     {
96         // updated
97         $book = Books::find($id);
98         $book->delete();
99         return redirect('books')->with('success','Book has been deleted');
00
01     }

```

- Now go to <http://localhost:8000/books> and delete one of the books. Also, check the corresponding entry in the database.

Step 11: Edit the books

- To perform edit, first write the corresponding code for edit() in BookController.php.

```

71     public function edit($id)
72     {
73         // updated
74         $books = Books::find($id);
75         return view('books.edit',compact('books','id'));
76     }

```


- Next in C:\wamp\www\lv1\SampleLv1\resources\views\books, create the corresponding view edit.blade.php file and write the following code in it.

```
1. <!-- edit.blade.php -->
2.
3. <!DOCTYPE html>
4. <html>
5.   <head>
6.     <meta charset="utf-8">
7.     <title>Laravel 5.5 CRUD Tutorial</title>
8.     <link rel="stylesheet" href="{{asset('css/app.css')}}">
9.   </head>
10.  <body>
11.    <div class="container">
12.      <h2>Edit A Book</h2><br />
13.      @if ($errors->any())
14.        <div class="alert alert-danger">
15.          <ul>
16.            @foreach ($errors->all() as $error)
17.              <li>{{ $error }}</li>
18.            @endforeach
19.          </ul>
20.        </div><br />
21.      @endif
22.      <form method="post" action="{{action('BookController@update', $id)}}">
23.        {{csrf_field()}}
24.        <input name="_method" type="hidden" value="PATCH">
25.        <div class="row">
26.          <div class="col-md-4"></div>
27.          <div class="form-group col-md-4">
28.            <label for="name">Name:</label>
29.            <input type="text" class="form-control" name="name" value="{{ $books->name }}">
30.          </div>
31.        </div>
32.        <div class="row">
33.          <div class="col-md-4"></div>
34.          <div class="form-group col-md-4">
35.            <label for="author">Author:</label>
36.            <input type="text" class="form-control" name="author" value="{{ $books->author }}">
37.          </div>
38.        </div>
39.        <div class="row">
40.          <div class="col-md-4"></div>
41.          <div class="form-group col-md-4">
42.            <label for="price">Price:</label>
43.            <input type="text" class="form-control" name="price" value="{{ $books->price }}">
```

```

44.     </div>
45.     </div>
46. </div>
47. <div class="row">
48.     <div class="col-md-4"></div>
49.     <div class="form-group col-md-4">
50.         <button type="submit" class="btn btn-success" style="margin-left:38px">Update
Book</button>
51.     </div>
52. </div>
53. </form>
54. </div>
55. </body>
56. </html>

```

- Next, write the following code for the update() in BookController.php.

```

85. public function update(Request $request, $id)
86. {
87.     // updated
88.     $book = Books::find($id);
89.     $this->validate(request(), [
90.         'name' => 'required',
91.         'author' => 'required',
92.         'price' => 'required|numeric'
93.     ]);
94.     $book->name = $request->get('name');
95.     $book->author = $request->get('author');
96.     $book->price = $request->get('price');
97.     $book->save();
98.     return redirect('books')->with('success', 'Book has been updated');
99. }

```

- Finally, update the corresponding book and view in webpage and in database.

Screenshots: (for editing and deletion)

ID	Name	Author	Price	Action
4	Modern Database Management	McFadden	400	Edit Delete
5	Database Systems: Principles, Design, and Implementation	Ricardo & Macmillan	1000	Edit Delete
6	Database Management Systems	Ramarkrishnan & Gehrke	850	Edit Delete

Edit A Book

Name:

Author:

Price:

Update Book

Book has been updated

ID	Name	Author	Price	Action
4	Modern Database Management	McFadden	600	Edit Delete
5	Database Systems: Principles, Design, and Implementation	Ricardo & Macmillan	1000	Edit Delete
6	Database Management Systems	Ramarkrishnan & Gehrke	850	Edit Delete

```
mysql> select * from books;
```

id	name	author	price	created_at
4	Modern Database Management	McFadden	600	2019-05-06 16
5	Database Systems: Principles, Design, and Implementation	Ricardo & Macmillan	1000	2019-05-06 16
6	Database Management Systems	Ramarkrishnan & Gehrke	850	2019-05-06 16

3 rows in set (0.00 sec)

-----Task 9.1-----

Complete the sample CRUD implemented in this lab and demonstrate to instructor.

-----Task 9.2-----

Answer the following questions:

1. What is a model? Specify the location of a model in Laravel Directory.
2. What is a view? Specify the location of a view in Laravel Directory.
3. For a given model e.g. Books, how many view files are created and why? State the purpose of each view file.
4. What is controller? Specify the location of a controller in Laravel Directory.
5. What is the purpose of following functions in controller file: index(), create(), store(), show(), edit(), update(), and destroy().
6. What is the purpose of routes.php or routes folder (i.e. web.php) in Laravel?

7. What is the meaning of following command: `php artisan route:list`? What does the generated table show?
8. What is the meaning of following command: `php artisan serve`? On what port it works on default? How the default port can be changed?
9. What is the purpose of `.env` file?
10. What happens if the database in step 2 is not created?

-----Task 9.3-----

Implement your DBMS Project in Laravel.

Registration #: _____

Name: _____

Date: _____

CSE 404L – Database Management Systems Lab

LAB ASSESSMENT RUBRICS

DBMS LAB 09 – Laravel MVC Framework

Dimension	Exemplary	Acceptable	Developing	Unsatisfactory	Student Score out of 10 Marks
	10	8	6	4	
Overall Impression of Lab Report	Report is complete, well written, and organized appropriately with additional elements that enhance it.	Report is complete, briefly written, and organized. Lacks additional elements.	Report is mostly complete, loosely written, and fairly organized.	Report is incomplete, sloppy, and/or disorganized.	
Submission	Report is submitted on time.	Report is submitted within 24 hours of due date.	Report is submitted within 72 hours of due date.	Report was more than 3 days overdue.	
Specification	Programs work and exceed specifications.	Programs work and meet all specifications.	Programs work and meet partial specifications.	Programs work but fail to meet any specification.	
Output Figures/Graphics	All the output figures and graphics are shown clearly and labeled.	Most output figures and graphics are shown clearly and labeled.	Few of the output figures and graphics are shown and labeled.	Output figures and graphics are not shown and not labeled.	
Verbal Communication and Understanding	Answered clearly and accurately with sufficient knowledge of Laravel MVC Framework.	Answered clearly and accurately with average knowledge of Laravel MVC Framework.	Answered somewhat clearly and somewhat accurately with limited knowledge of Laravel MVC Framework.	Answered wrongly and inaccurately with no knowledge of Laravel MVC Framework..	

Marks: (_____ + _____ + _____ + _____ + _____)/5 = _____

Teacher Remarks and Signature: _____
