Lab # 10: DBMS LAB Project – Laravel Implementation [OEL]

OBJECTIVES OF THE LAB

This lab covers fourth part of DBMS Lab Project Submission that is Laravel Implementation.

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

PHP MVC FRAMEWORK IN LARAVEL

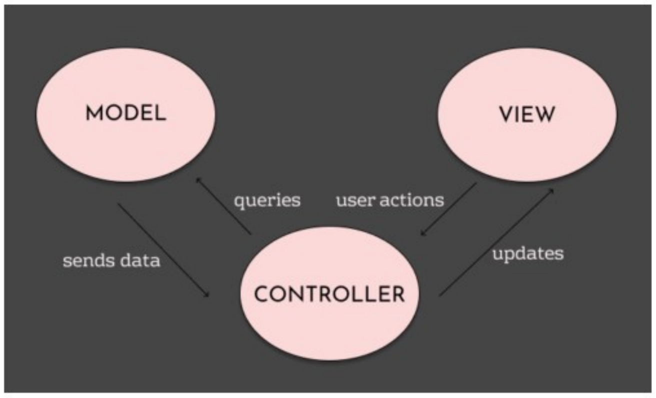


Figure 9.1 – Model-View-Controller Framework

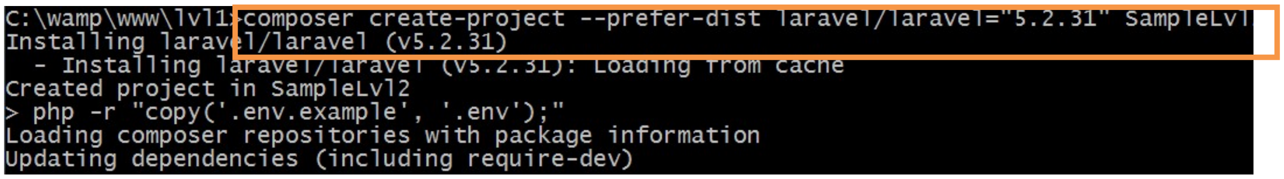
Figure 9.1 depicts the MVC Framework where M stands for Model, V stands for View and C stands for Controller. A **Model** is a representation of a real-life instance or object in our code base. It can either be a database or JSON File or some other source. The **View** represents the user interface through 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

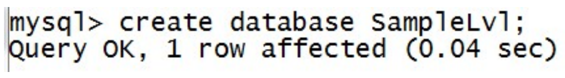
This example uses MySQL DBMS and PDO PHP Extension.

Step 1: Installing the Laravel 5.2 Framework.



```
C:\wamp\www\lvl1>composer create-project --prefer-dist laravel/laravel="5.2.31" SampleLvl
Installing laravel/laravel (v5.2.31)
  - Installing laravel/laravel (v5.2.31): Loading from cache
Created project in SampleLvl2
> php -r "copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies (including require-dev)
```

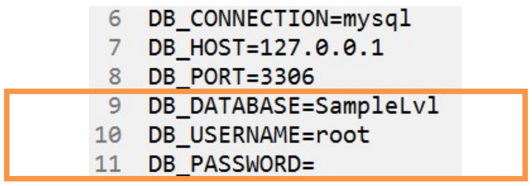
Step 2: Create the database.



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

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

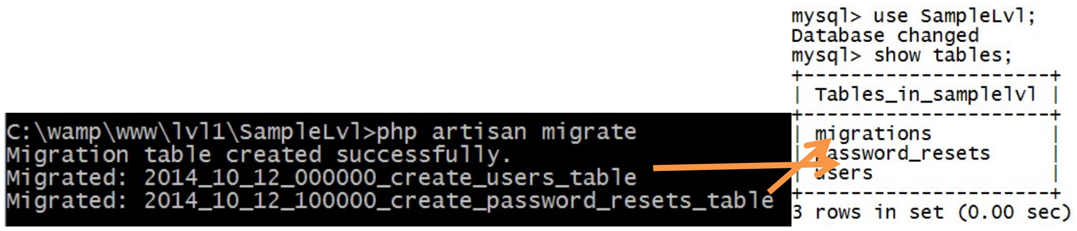
- Open file: C:\wamp\www\lvl1\SampleLvl\.env and do following changes:



```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=SampleLvl
DB_USERNAME=root
DB_PASSWORD=
```

- Save the file.

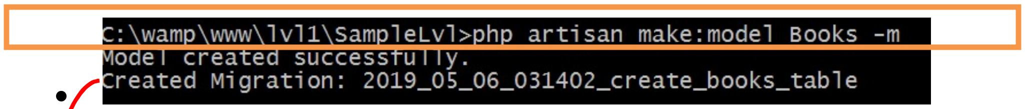
Step 4: Migrate the build-in tables in our database.



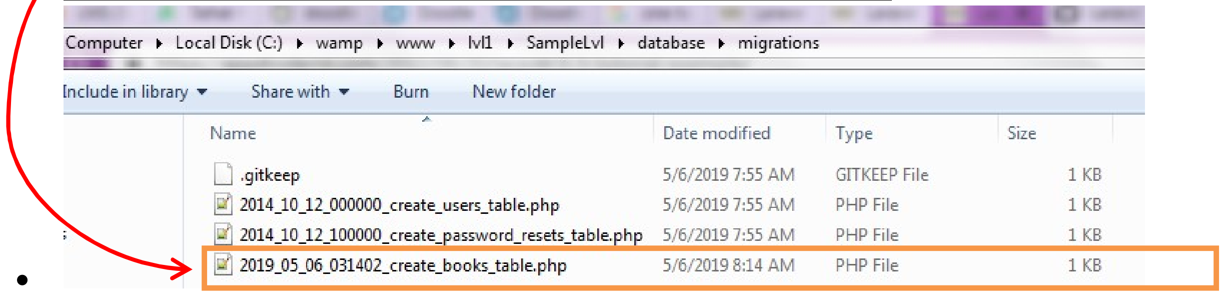
```
C:\wamp\www\lvl1\SampleLvl>php artisan migrate
Migration table created successfully.
Migrated: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_100000_create_password_resets_table
```

```
mysql> use SampleLvl;
Database changed
mysql> show tables;
+----------------------+
| Tables_in_samplelvl  |
+----------------------+
| 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.



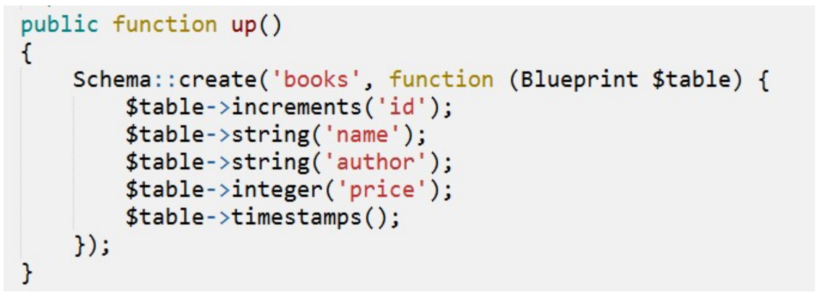
```
C:\wamp\www\lvl1\SampleLvl>php artisan make:model Books -m
Model created successfully.
Created Migration: 2019_05_06_031402_create_books_table
```



Computer ▸ Local Disk (C:) ▸ wamp ▸ www ▸ lvl1 ▸ SampleLvl ▸ database ▸ migrations
Include in library ▾
Share with ▾
Burn
New folder

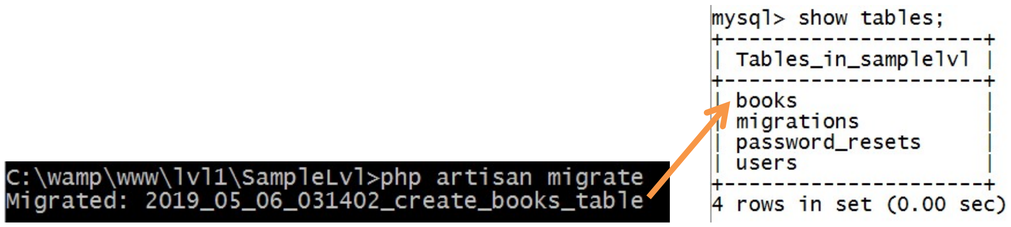
Name	Date modified	Type	Size
.gitkeep	5/6/2019 7:55 AM	GITKEEP File	1 KB
2014_10_12_000000_create_users_table.php	5/6/2019 7:55 AM	PHP File	1 KB
2014_10_12_100000_create_password_resets_table.php	5/6/2019 7:55 AM	PHP File	1 KB
2019_05_06_031402_create_books_table.php	5/6/2019 8:14 AM	PHP File	1 KB

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



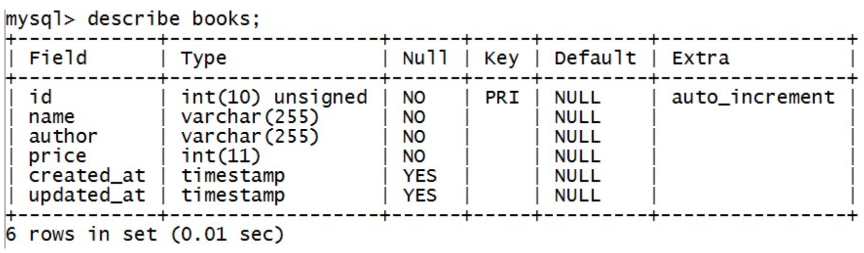
```php
public function up()
{
    Schema::create('books', function (Blueprint $table) {
        $table->increments('id');
        $table->string('name');
        $table->string('author');
        $table->integer('price');
        $table->timestamps();
    });
}
```

- 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\lvl1\SampleLvl>php artisan migrate
Migrated:  2019_05_06_031402_create_books_table
```

```
mysql> show tables;
+---------------------+
| Tables_in_samplelvl |
+---------------------+
| 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.

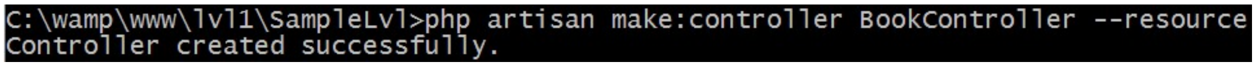
- Go to C:\wamp\www\lvl1\SampleLvl\resources\views. Create folder for the books model. Name it books. Go inside the newly created folder.
- Create the file: **create.blade.php**. Add the following code in it:

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

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

```
            <div class="form-group col-md-4">
            <label for="name">Name:</label>
            <input type="text" class="form-control" name="name">
            </div>
        </div>
        <div class="row">
            <div class="col-md-4"></div>
            <div class="form-group col-md-4">
            <label for="name">Author:</label>
            <input type="text" class="form-control" name="author">
            </div>
        </div>
        <div class="row">
            <div class="col-md-4"></div>
            <div class="form-group col-md-4">
            <label for="price">Price:</label>
            <input type="text" class="form-control" name="price">
            </div>
        </div>
    </div>
        <div class="row">
            <div class="col-md-4"></div>
            <div class="form-group col-md-4">
            <button type="submit" class="btn btn-success" style="margin-left:38px">Add Book</button>
            </div>
        </div>
        </form>
    </div>
</body>
</html>
```

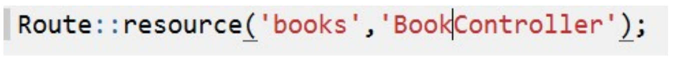
- Go to following folder: C:\wamp\www\lvl1\SampleLvl\public. Create the folder css in it and store the given app.css file in it.

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

- 

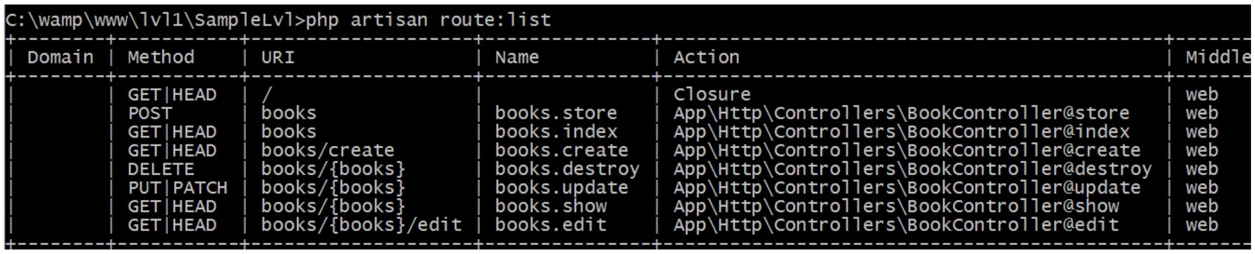
```
C:\wamp\www\lvl1\SampleLvl>php artisan make:controller BookController --resource
Controller created successfully.
```

- It will generate one controller file called BookController.php in C:\wamp\www\lvl1\SampleLvl\app\Http\Controllers.
- Next, in the routes configuration file — C:\wamp\www\lvl1\SampleLvl\app\Http\routes.php— add the following to define a Books resource route:



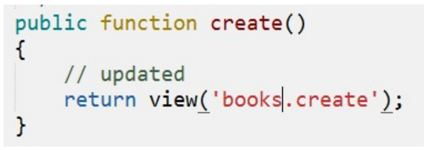
```
Route::resource('books','BookController');
```

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



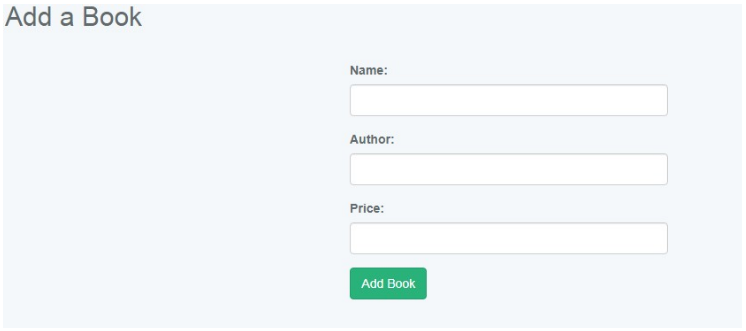
```
C:\wamp\www\lvl1\SampleLvl>php artisan route:list
+--------+-----------+---------------------+---------------+-----------------------------------------------+-------
| Domain | Method    | URI                 | Name          | Action                                        | Middle
+--------+-----------+---------------------+---------------+-----------------------------------------------+-------
|        | 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.



```php
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

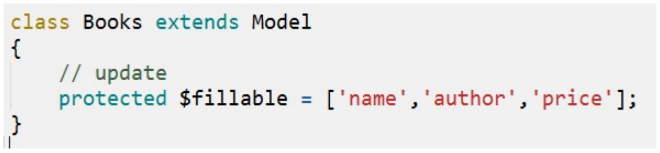
Step 8: Put the Laravel 5.5 Validation in Product Form.

- Open the C:\wamp\www\lvl1\SampleLvl\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')}}">
```

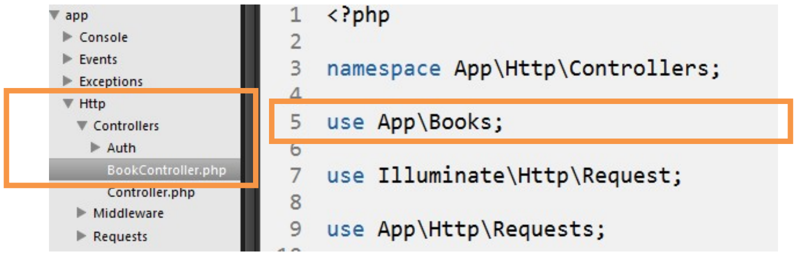
```
{{csrf_field()}}
```

- Next, go to C:\wamp\www\lvl1\SampleLvl\app\Books.php. Add the following code:



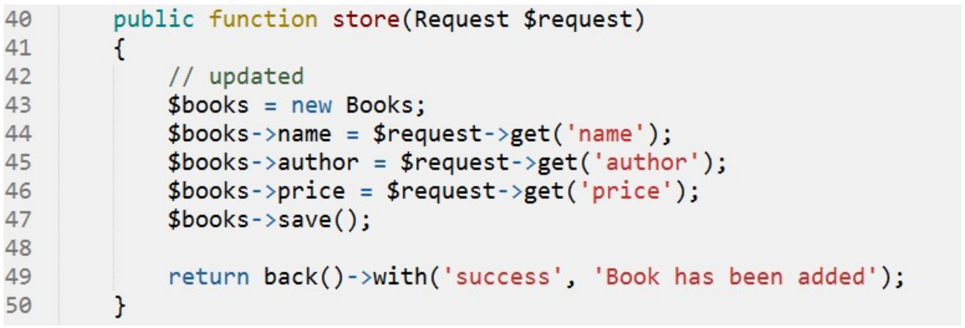
```
class Books extends Model
{
    // update
    protected $fillable = ['name','author','price'];
}
```

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



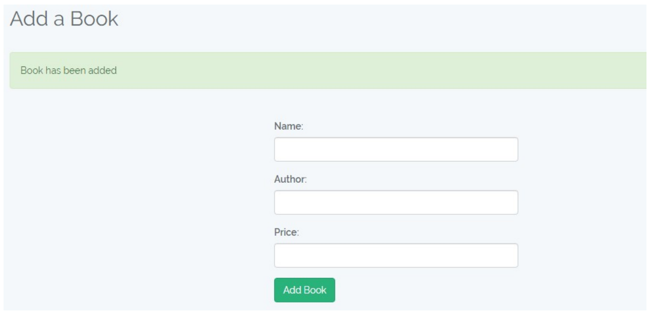
```
<?php

namespace App\Http\Controllers;

use App\Books;

use Illuminate\Http\Request;

use App\Http\Requests;
```

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

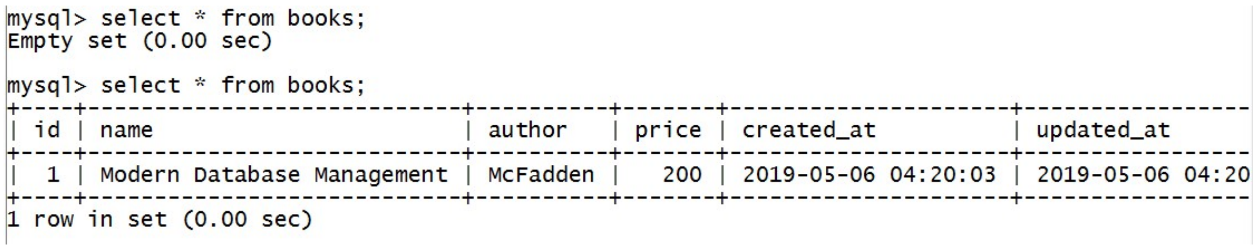


```
    public function store(Request $request)
    {
        // updated
        $books = new Books;
        $books->name = $request->get('name');
        $books->author = $request->get('author');
        $books->price = $request->get('price');
        $books->save();

        return back()->with('success', 'Book has been added');
    }
```

- 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.

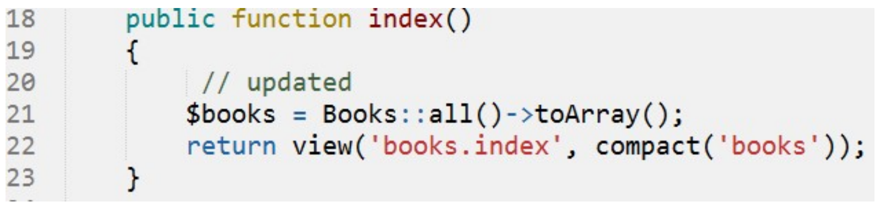


- Check in database.



```
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
+----+----------------------------+----------+-------+---------------------+--------------------
1 row in set (0.00 sec)
```

Step 9: Make an index page to list the books

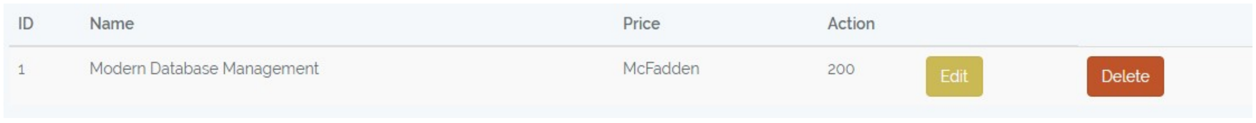


```php
    public function index()
    {
        // updated
        $books = Books::all()->toArray();
        return view('books.index', compact('books'));
    }
```

- In C:\wamp\www\lvl1\SampleLvl\resources\views\books, create one blade file called index.blade.php file and put the following code in it.

```html
<!-- index.blade.php -->

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Index Page</title>
    <link rel="stylesheet" href="{{asset('css/app.css')}}">
  </head>
  <body>
    <div class="container">
    <br />
    @if (\Session::has('success'))
      <div class="alert alert-success">
        <p>{{ \Session::get('success') }}</p>
      </div><br />
     @endif
    <table class="table table-striped">
    <thead>
      <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Author</th>
        <th>Price</th>
        <th colspan="2">Action</th>
      </tr>
    </thead>
    <tbody>
      @foreach($books as $book)
      <tr>
        <td>{{$book['id']}}</td>
        <td>{{$book['name']}}</td>
        <td>{{$book['author']}}</td>
        <td>{{$book['price']}}</td>
        <td><a href="{{action('BookController@edit', $book['id'])}}" class="btn btn-warning">Edit</a></td>
        <td>
          <form action="{{action('BookController@destroy', $book['id'])}}" method="post">
            {{csrf_field()}}
            <input name="_method" type="hidden" value="DELETE">
            <button class="btn btn-danger" type="submit">Delete</button>
          </form>
        </td>
      </tr>
      @endforeach
    </tbody>
  </table>
  </div>
  </body>
</html>
```

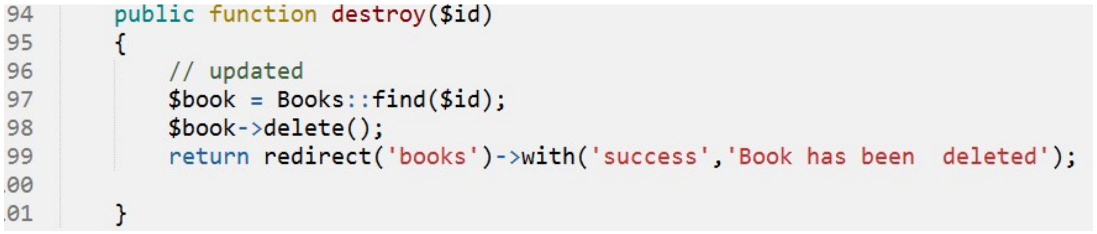
- Now go to following: http://localhost:8000/books. Books are listed.



ID	Name	Price	Action		
1	Modern Database Management	McFadden	200	Edit	Delete

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.

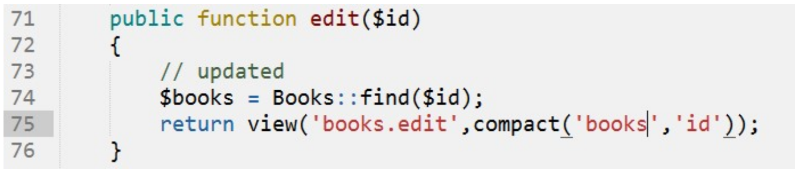


```
    public function destroy($id)
    {
        // updated
        $book = Books::find($id);
        $book->delete();
        return redirect('books')->with('success','Book has been  deleted');

    }
```

- 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.

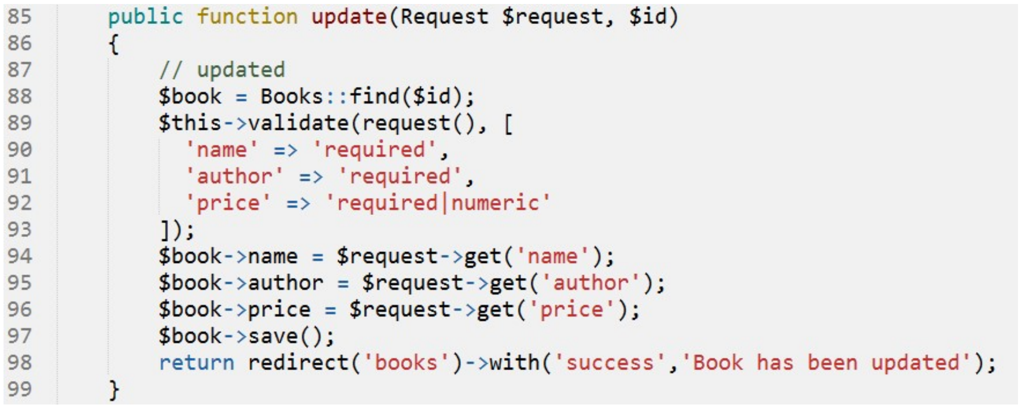


```
    public function edit($id)
    {
        // updated
        $books = Books::find($id);
        return view('books.edit',compact('books','id'));
    }
```

- Next in C:\wamp\www\lvl1\SampleLvl\resources\views\books, create the corresponding view edit.blade.php file and write the following code in it.

```
<!-- edit.blade.php -->

<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Laravel 5.5 CRUD Tutorial</title>
  <link rel="stylesheet" href="{{asset('css/app.css')}}">
 </head>
 <body>
  <div class="container">
   <h2>Edit A Book</h2><br  />
   @if ($errors->any())
   <div class="alert alert-danger">
     <ul>
       @foreach ($errors->all() as $error)
         <li>{{ $error }}</li>
       @endforeach
     </ul>
   </div><br />
   @endif
   <form method="post" action="{{action('BookController@update', $id)}}">
    {{csrf_field()}}
    <input name="_method" type="hidden" value="PATCH">
    <div class="row">
     <div class="col-md-4"></div>
     <div class="form-group col-md-4">
      <label for="name">Name:</label>
      <input type="text" class="form-control" name="name" value="{{$books->name}}">
     </div>
    </div>
    <div class="row">
     <div class="col-md-4"></div>
      <div class="form-group col-md-4">
       <label for="author">Author:</label>
       <input type="text" class="form-control" name="author" value="{{$books->author}}">
      </div>
     </div>
    <div class="row">
     <div class="col-md-4"></div>
      <div class="form-group col-md-4">
       <label for="price">Price:</label>
       <input type="text" class="form-control" name="price" value="{{$books->price}}">
      </div>
     </div>
    </div>
    <div class="row">
     <div class="col-md-4"></div>
     <div class="form-group col-md-4">
      <button type="submit" class="btn btn-success" style="margin-left:38px">Update Book</button>
     </div>
    </div>
   </form>
  </div>
 </body>
</html>
```

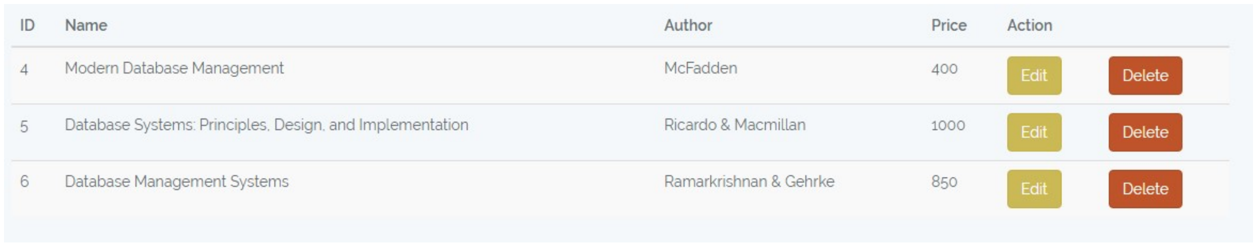
- Next, write the following code for the update() in BookController.php.



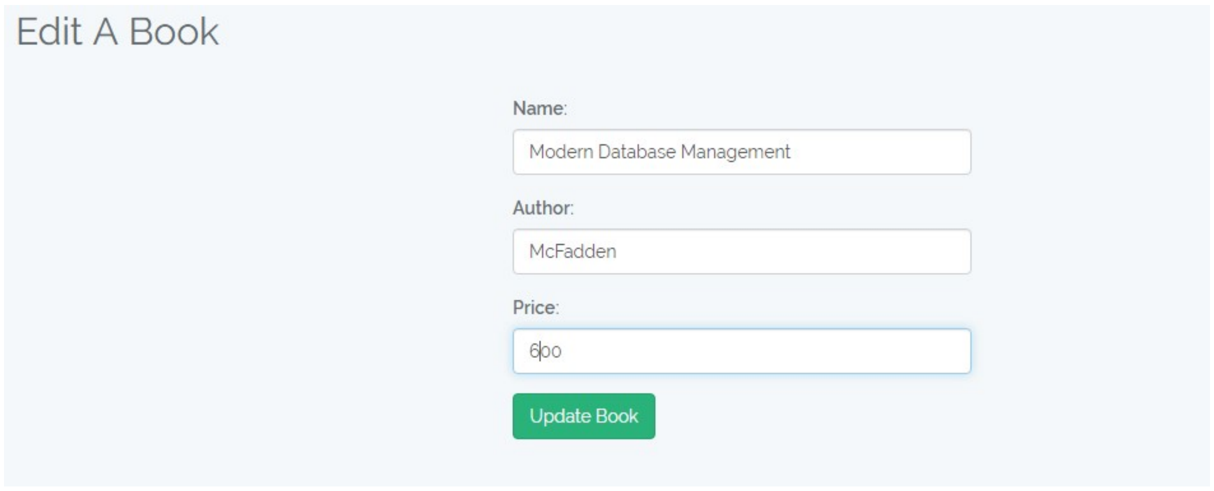
```
    public function update(Request $request, $id)
    {
        // updated
        $book = Books::find($id);
        $this->validate(request(), [
          'name' => 'required',
          'author' => 'required',
          'price' => 'required|numeric'
        ]);
        $book->name = $request->get('name');
        $book->author = $request->get('author');
        $book->price = $request->get('price');
        $book->save();
        return redirect('books')->with('success','Book has been updated');
    }
```

-
- 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:

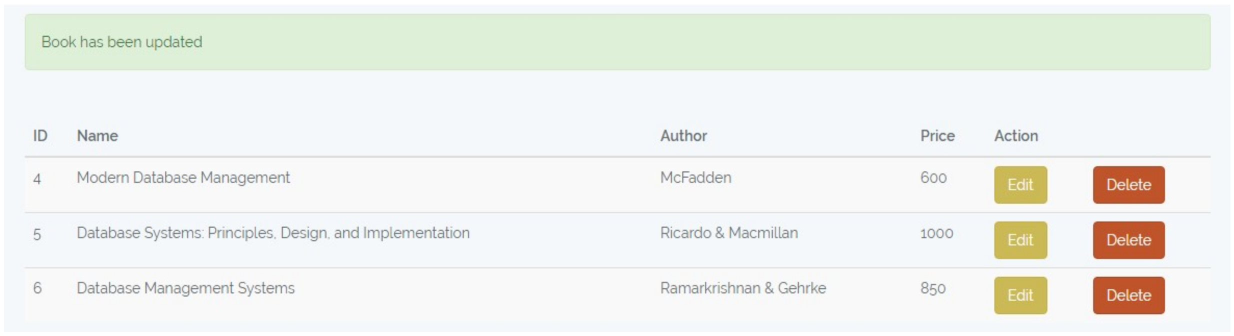
Modern Database Management

Author:

McFadden

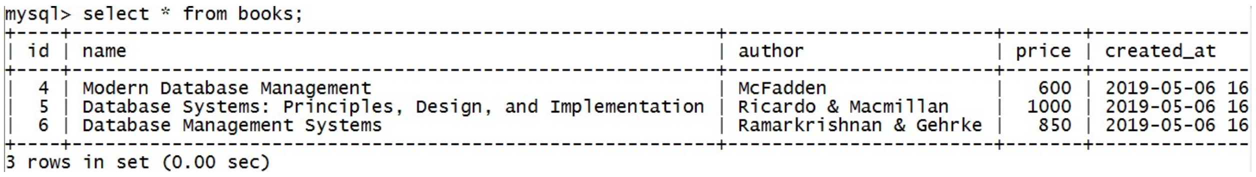
Price:

600



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 10.1--------------------------

The objective of open-ended laboratories is to foster problem-solving skills, critical thinking, creativity, and engineering hands-on experience. When developing solutions, you are frequently required to consider multiple factors, such as design constraints, cost-effectiveness, sustainability, and safety. To achieve the desired results, they may need to conduct research, conduct experiments, analyze data, and iterate your designs.

Since this is an open-ended lab, you can choose the framework (e.g., Laravel, MERN, Python Django, etc.) based on your inclinations and develop an original solution using your creativity and problem-solving skills. Independent thought, investigation, and experimentation are emphasized.