Laravel Lab Command

1. Create a project

```
composer create-project laravel/laravel example-app
```

2. After creating the project go to .env file and change **DB_CONNECTION=mysql**
3. Also create a database and run the following command. It **deletes all the existing tables of the database and runs the migrate command**.

```
php artisan migrate:fresh
```

4. The command `php artisan | grep 'make'` is used to list all Artisan commands that contain the word "make."
5. Create model, controller, migration

```
php artisan make:model Book -cfm
```

6. Now create the Book database add necessary column and run the following command:

```
php artisan migrate
```

> Putting fake data into the database is called seeding.
>
> And for creating fake data we use Faker library of the PHP.

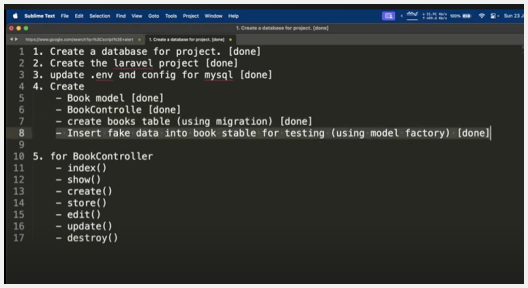
7. Now after defining the `Book Factory` we will go to the seeder file and generate the number of fake data using this command.

```
php artisan db:seed
```

You can use truncate command every time for empty the database table then initialize them with new value.

`Book::truncate();`

Till now we have completed up-to this portion.



8. In MVC we have router and that router maintain a routing table. We will write all the router in `router/web.php` file and those router are responsible for controlling the router.

- Now go to the `BookController.php` file and write a index() function.
- After that go to the `web.php` file and define the router. This code representing that if you go to the root folder then you will get the `BookController` and call the `index` method. And this method will call a view with all the book information.

```
Route::get('/', [BookController::class, 'index']);
```

`BookController.php` has this content:

```
class BookController extends Controller
{
```

```
    public function index(){
        $books = Book::all();
        return view('books.index')//got to book/index file
        ->with('books', $books);
    }
}
```

And in the `Book/view/index.blade.php` we have this:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-s
    <title>Document</title>
</head>
<body>
    <h1>Hello Laravel</h1>

    <ol>
        @foreach($books as $book)
            <li>{{$book->title}} Tk. ({{$book->price}})</li>
        @endforeach

    </ol>
</body>
</html>
```

9. Now you have to design the view by creating a table. Also you need to set up a router for showing a specific book details. Remember when you are putting curly braces `{}` it representing a parameter you are passing as an URL.

```
Route::get('/{id}/show', [BookController::class, 'show']);
```

10. Now what you have to do whenever you click on the `Details` URL button it will show up a separate page and show the details of the book. Also in the route we have use named route which enable us to change the raw route as we like.
11. Before moving on there is a problem the root page is showing all the books. So we have to add `pagination`. But the problem is by default it is compatible with tailwind CSS.

```
{{$books->links()}}
```

So, we convert it to `bootstrap` using this in `Providers/AppServiceProvider.php`

```
use Illuminate\Pagination\paginator;
public function boot(): void
    {
        Paginator::useBootstrapFive();
      }
```

12. Now there is a problem we use bootstrap for decorating the page. But we have to do that for each page. For solving that we can use layout. So create a layout in `views/layout.blade.php` and there write this in the body section:

    ```
    @yield('page-content');
    ```

Now In the other view page we just add the content and this will load the content in the layout page.

```
@extends('layout')
@section('page-content')
```

13. Now add a button to add new entry. After adding the button whenever we click the button it will create a form and there we will add necessary information and insert the book. For adding information we will create a view and we can also create a view using command line.

```
php artisan make:view books.create
```

So this view page will contain the view of adding new page. Now the question is how we going to come to that page for that we have to define a route:

```
Route::get('books/create', [BookController::class, 'create'])->
```

Now we add a forms in the `create.blade.php` . So, for showing the form we have use a create method now we have to store the information of the form. For that reason we have to use a `Store()` method.

And for storing the information we have to define a route and that will be a post method.

```
Route::post('/books',[BookController::class, 'store'])->name('b
```

and in the `BookController` store method will look like this

```
public function store(Request $request){
        dd($request->all());//will show the data.
    }
```

and the form will look like this

```
<form method="post" action="{{route('books>.store')}}">

    {{csrf_field()}}
```

here `{{csrf_field()}}` will create a token for preventing cross site request forgery attack.

Now we want to store the data into the database.

```
public function store(Request $request){
        Book::create($request->all());//mass assign value.
```

```
        return redirect()->route('books.index');
    }
```

When you try to assign value this it will show and error. For making it write you have to got to `Models/Book.php` and add this. This will allow you to mass assign:

```
protected $fillable = [
        'title',
        'author',
        'isbn',
        'price',
        'stock'

    ];
```

14. Now we have to add validation for that you do this in the `BookController`

```
public function store(Request $request){
        $rules = [
            'title' => 'required',
            'author' => 'required',
            'isbn' => 'required|size:13',
            'stock' => 'required|numeric|integer|gte:0',
            'price' => 'required|numeric'
        ];
        $request->validate($rules);

        Book::create($request->all());
        return redirect()->route('books.index');
    }
```

But you need to see the type of error for that you go to the `create.blade.php` file and add this line under each input field.

```
<div>{{$errors->first('stock')}}</div>
```

But one problem still remains when you make some mistake it will show that error and the value of that field get vanish and redirect to the same page. For preventing that we can do this:

```
<input type="text" class="form-control" name="title" value="{{o
```

For creating custom validation message we can do this:

```
public function store(Request $request){
        $rules = [
            'title' => 'required',
            'author' => 'required',
            'isbn' => 'required|size:13',
            'stock' => 'required|numeric|integer|gte:0',
            'price' => 'required|numeric'
        ];
        $message=[
            'stock.gte' => 'The stock must be grater than or equ
        ];
        $request->validate($rules, $message);

        $book = Book::create($request->all());
        return redirect()->route('books.show', $book->id);
    }
```

15. Now we want to add delete method. And we want to execute delete in the post method for that in the router we have to do this :

```
Route::delete('/books/{id}', [BookController::class, 'destroy']
```

We also have to add delete button inside `index.blade.php` like this

```
<td>
            <a href="{{route('books.show', $book->id)}}">Details
```

```
            <form method="post" action="{{route('books.destroy',
                @csrf // needed for csrf attack
                @method('DELETE')// method spoofing cz browser u
                <input class="btn btn-link" type="submit" value=
            </form>

        </td>
```

and in the `BookController` the code will be

```
public function destroy(Request $request, $id){
        $book = Book::find($id);
        $book->delete();
        return redirect()->route('books.index');
    }
```

16. Now for edit we have to define the route then a method. After that we have to define the view.
    Because first of all you have to edit than update that value lets see them sequentially.

```
Route::get('books/{id}/edit', [BookController::class, 'edit'])->
Route::post('books/update', [BookController::class, 'update'])->
```

In `BookController` you have to add `edit` & `Update`

```
public function edit($id){
        $book = Book::find($id);
        return view('books.edit')
        ->with('book', $book);
    }

    public function update(Request $request){
        $rules = [
            'title' => 'required',
```

```
            'author' => 'required',
            'isbn' => 'required|size:13',
            'stock' => 'required|numeric|integer|gte:0',
            'price' => 'required|numeric'
        ];
        $message = [
            'stock.gte' => 'The stock must be grater than or equ
        ];
        $request->validate($rules, $message);

        $book = Book::find($request->id);
        $book->title = $request->title;
        $book->author = $request->author;
        $book->isbn = $request->isbn;
        $book->stock = $request->stock;
        $book->price = $request->price;
        $book->save();

        return redirect()->route('books.show',$book->id)->with(
    }
```

Then in the view create similar view just like `create.blade.php` but the routing path just different.

17. Finally for search functionality we will create a search bar and search button then in the `BookController` we will create a logic if search parameter exist then we will execute that block else we will show up all the books list.

```
public function index(Request $request)
    {
        if ($request->has('search')) {
            $books = Book::where('title', 'like', '%' . $request
                ->orWhere('author', 'like', '%' . $request->sear
                ->paginate(10);
        } else {
            $books = Book::paginate(50);  // This should be in a
```

```
        }

        return view('books.index')
        ->with('books', $books);
    }
```