Laravel BookShop Project – Step-by-Step Setup Notes

Step 1: Create Database

Command:

```
CREATE DATABASE laravel_bookshop;
```

Why:
Laravel requires a MySQL database to store data such as books, users, and orders.

Step 2: Create a New Laravel Project

Commands:

```
composer create-project laravel/laravel laravel-bookshop
cd laravel-bookshop
php artisan serve
```

Why:
Sets up a fresh Laravel application with the default folder structure.

✅ Check:
Visit http://127.0.0.1:8000 — you should see the Laravel welcome page.

Step 3: Configure Database Connection

Edit `.env` file:

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

Then run:

```
php artisan config:clear
php artisan migrate
```

Why:
This tells Laravel how to connect to your MySQL database and initializes the default migrations (users, password resets, etc.).

Step 4: Create Book Model, Controller & Factory

Command:

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

Breakdown:

Model (`Book`) → Represents the `books` table.

Controller (`BookController`) → Handles all book-related logic (CRUD operations).

Factory → Generates fake book data for testing.

Migration → Defines the structure of the `books` table.

Step 5: Define Books Table Structure (Migration)

Edit file:
`database/migrations/xxxx_xx_xx_create_books_table.php`

Code:

```
public function up(): void
{
    Schema::create('books', function
(Blueprint $table) {
        $table->id();
        $table->string('title', 255);
        $table->string('author');
        $table->string('isbn', 13);
        $table->smallInteger('stock')-
>default(0);
        $table->float('price', 8, 2);
        $table->timestamps();
    });
}
```

Run migration:

```
php artisan migrate:fresh
```

Why:
Rebuilds all tables using the latest migration definitions.

Step 6: Insert Fake Data Using Model Factory

✓ Edit Factory

File: `database/factories/BookFactory.php`

```
return [
    'title'  => $this->faker->sentence,
    'author' => $this->faker->name,
    'isbn'   => $this->faker->unique()->isbn13(),
    'stock'  => $this->faker->numberBetween(0, 50),
    'price'  => $this->faker->randomFloat(2, 10, 100),
];
```

✓ Edit Seeder

File: `database/seeders/DatabaseSeeder.php`

```
use App\Models\Book;

\App\Models\User::truncate();
\App\Models\Book::truncate();
\App\Models\Book::factory(100)->create();
```

Run:

```
php artisan db:seed
```

Why:
Populates the database with 100 fake books for testing and UI development.

Verify Data

Check in MySQL or phpMyAdmin:

```
SELECT * FROM books;
```

Check:
You should see 100 fake books generated by the factory.

Step 7: Display Books on the Web Page

1. Create View File

Path:
`resources/views/books/index.blade.php`

Code:

```
<h1>Book List</h1>

<ol>
    @foreach ($books as $book)
        <li>{{ $book->title }} by {{ $book-
>author }}</li>
    @endforeach
</ol>
```

Why:
Displays all books fetched from the database using a simple Blade template.

---

2. Update Controller

Path:
`app/Http/Controllers/BookController.php`

Code:

```
<?php

namespace App\Http\Controllers;

use App\Models\Book;
use Illuminate\Http\Request;

class BookController extends Controller
{
    public function index()
    {
        $books = Book::all();
        return view('books.index')-
>with('books', $books);
    }
}
```

Why:
The `index()` method retrieves all books from the `books` table and passes them to the view.

---

3. Define Route

Path:
`routes/web.php`

Code:

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\BookController;

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

Why:
This route sends requests from the home URL (/) to the `BookController@index` method.

https://laravel.com/docs/12.x/pagination#cursor-paginator-instance-methods

---

Step 1 — Enable Bootstrap 5 for Pagination

File: app/Providers/AppServiceProvider.php

1. Add:
use Illuminate\Pagination\Paginator;

2. Inside boot() method:
Paginator::useBootstrapFive();

Step 2 — Controller Pagination

In your controller (e.g. BookController):

```
public function index()
{
   $books = Book::paginate(10);
   return view('books.index')->with('books', $books);
}
```

Step 3 — Blade Pagination Links

At the bottom of your index.blade.php, add:

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

Step 4 — Layout Setup

In resources/views/layout.blade.php, add:

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

Use this in each view:

```
@extends('layout')
@section('page-content')
   <!-- Page content here -->
@endsection
```

Step 5 — Table Design

In index.blade.php and show.blade.php:

```
<table class="table table-striped table-bordered">
   ...
</table>
```

Step 6 — Back Button in Show Page

At the bottom of show.blade.php:

```
<a href="{{ route('home') }}" class="btn btn-
primary">Back</a>
```

Step 7 — Optional Reference

Bootstrap examples:
https://getbootstrap.com/docs/5.3/examples/

In index page add

```
<a href = "{{route('show', $book->id)}}"> View</a>
```

In web.php

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

In bookcontroler

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

Steps for Adding a New Book in Laravel Project

1 Add “Add New Book” Button

Add New Book

2 Create Route for the Create Page

In web.php:

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

```
>name('books.create');
```

3 Add Create Function in Controller

In BookController.php:

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

4 Create Page Setup

In books/create.blade.php, add the form.

Add CSRF Protection:

```
@csrf
```

Use form layout from Bootstrap (Reference: https://getbootstrap.com/docs/5.3/forms/overview/#overview)

```
<div class="mb-3">
   <label for="title" class="form-label">Title</label>
   <input type="text" class="form-control" id="title"
name="title" value="{{old('title')}}">
   <div class="text-danger">{{$errors->first('title')}}</div>
</div>
```

5 Store Data (Form Submission Route)

In web.php:

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

6 Add Required Imports in Controller

In BookController.php:

```
use Illuminate\Http\Request;
```

In create page:

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

7 Update Book Model

In Book.php:

```
class Book extends Model
{
   /** @use HasFactory<\Database\Factories\BookFactory> */
   use HasFactory;

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

8 Add Validation and Store Logic

Reference: https://laravel.com/docs/12.x/validation#quick-writing-the-validation-logic

```
public function store(Request $request)
{
   $rules = [
      'title' => 'required|max:255',
      'author' => 'required|max:255',
      'isbn' => 'required|size:13',
      'stock' => 'required|numeric|integer',
      'price' => 'required|numeric'
   ];

   $messages = [
      'isbn.size' => 'The ISBN must be exactly 13 characters.',
      'stock.integer' => 'The stock must be an integer value.',
   ];

   $request->validate($rules, $messages);

   Book::create($request->all());
   return redirect()->route('home')->with('success', 'Book created successfully.');
}
```

✅ Now your “Add New Book” feature is complete. Visit /books/create to test.

Steps for Deleting a Book in Laravel Project

1 Add Route for Deleting a Book

In `web.php`:

```
Route::delete('/books/{id}', [BookController::class, 'delete'])->name('books.delete');
```

---

2 Add Delete Function in Controller

In `BookController.php`:

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

---

3 Add Delete Button/Form in Index Page

In `index.blade.php`:

```
<form method="post" action="{{ route('books.delete', $book->id) }}">
    @csrf
```

@method('delete')

<button type="submit" class="btn btn-link text-danger">Delete</button></form>

Steps to Implement Edit Functionality

Step 1: Add Edit Route in web.php

Add the following route to enable the edit functionality for each book:

Route::get('books/{id}/edit', [BookController::class, 'edit'])->name('books.edit');

Step 2: Add edit() Method in BookController.php

The edit() method fetches the book data by its ID and sends it to the edit view.

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

}

Step 3: Create edit.blade.php Page

Create a new file named edit.blade.php in resources/views/books/ directory.
This page will be similar to create.blade.php but will show existing book details.

```
<input type="hidden" name="id" value="{{ $book->id }}">
```

-
```
<div class="mb-3">
  <label for="title" class="form-label">Title</label>
  <input type="text" class="form-control" id="title"
name="title"
      value ="{{ old('title', $book->title) }}">
  <div class="text-danger">{{ $errors->first('title') }}</div>
</div>
```

Step 4: Add Update Route in web.php

This route handles the form submission to update the book data.

-
```
Route::post('books/update', [BookController::class,
'update'])->name('books.update');
```

Step 5: Add update() Method in BookController.php

The update() method validates and updates the book record in the database.

-

- public function update(Request $request)
  {
  $rules = [
  'title' => 'required|max:255',
  'author' => 'required|max:255',
  'isbn' => 'required|size:13',
  'stock' => 'required|numeric|integer',
  'price' => 'required|numeric'
  ];

  $messages = [
  'isbn.size' => 'The ISBN must be exactly 13 characters.',
  'stock.integer' => 'The stock must be an integer value.',
  ];

  $request->validate($rules, $messages);

  $book = Book::find($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('home')->with('success', 'Book
updated successfully.');
}
```

Step 6: Add Edit Button in books.index.blade.php

Add an Edit button beside each book record to navigate to the edit page.

```
<a href="{{ route('books.edit', $book->id) }}" class="btn btn-
sm btn-warning">Edit</a>
```

Steps to Implement Search Functionality

Step 1: Add Search Form in index.blade.php

Add the following Bootstrap search form at the top of your book listing page (index.blade.php).
It allows users to search books by title or author.

```
<div class="row mb-3 align-items-center">
  <!-- Search Section -->
  <div class="col-lg-10 bg-light p-3">
    <form method="get" action="{{ route('home') }}">
      <div class="row g-2">
        <div class="col-8">
          <input type="text" class="form-control"
```

```
name="search" placeholder="Search by Title or Author"
                    value="{{ request('search') }}">
            </div>
            <div class="col-auto">
              <button type="submit" class="btn btn-
secondary">Search</button>
            </div>
          </div>
      </form>
  </div>

  <!-- Add New Book Button -->
  <div class="col-lg-2 text-end mt-3 mt-lg-0">
     <a class="btn btn-primary"
href="{{ route('books.create') }}">Add New Book</a>
  </div>
</div>
```

Step 2: Modify index() Method in BookController.php

Update the index() method in BookController.php to handle search functionality.
This will allow users to filter books by title or author.

```
public function index(Request $request)
{
  $query = Book::query();

  if ($request->filled('search')) {
     $search = $request->search;
```

```
        $query->where('title', 'like', "%{$search}%")
              ->orWhere('author', 'like', "%{$search}%");
    }

    $books = $query->paginate(10);
    return view('books.index', compact('books'));
}
```