



Develop a Simple Laravel CRUD Application | December 3, 2024 | 11:47:51 PM

Objective

To develop a Laravel application that allows users to manage a list of books. The application must implement the following functionalities:

- Search
- Create
- Edit
- Update
- Delete

Application Description

You will create a Book Management System where users can perform CRUD operations on books. Each book will have the following attributes:

- Title (string)
- Author (string)
- Description (text)
- Published Year (integer)
- Genre (string)

Step-by-Step Instructions

Setup the Laravel Project

1. Install a new Laravel project:

```
C:\WINDOWS\system32\cmd.  X  +  v
Microsoft Windows [Version 10.0.22631.4541]
(c) Microsoft Corporation. All rights reserved.

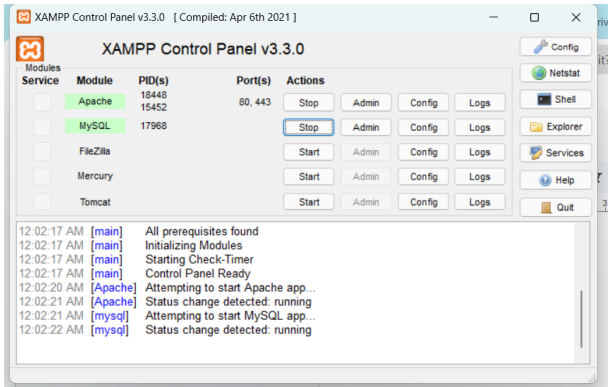
C:\Users\Administrator>cd \

C:\>laravel new book-management
```

```
Would you like to install a starter kit? [No starter kit]:
[none    ] No starter kit
[breeze  ] Laravel Breeze
[jetstream] Laravel Jetstream
> none|
```

```
Which testing framework do you prefer? [Pest]:
[0] Pest
[1] PHPUnit
> 1|
```

Start Apache and MySQL on XAMPP



```
- Installing sebastian/cli-parser (3.0.2): Extracting archive
- Installing phpunit/php-timer (7.0.1): Extracting archive
- Installing phpunit/php-text-template (4.0.1): Extracting archive
- Installing phpunit/php-invoker (5.0.1): Extracting archive
- Installing phpunit/php-file-iterator (5.1.0): Extracting archive
- Installing theseer/tokenizer (1.2.3): Extracting archive
- Installing sebastian/lines-of-code (3.0.1): Extracting archive
- Installing sebastian/complexity (4.0.1): Extracting archive
- Installing sebastian/code-unit-reverse-lookup (4.0.1): Extracting archive
- Installing phpunit/php-code-coverage (11.0.7): Extracting archive
- Installing phar-io/version (3.2.1): Extracting archive
- Installing phar-io/manifest (2.0.4): Extracting archive
- Installing myclabs/deep-copy (1.12.1): Extracting archive
- Installing phpunit/phpunit (11.4.4): Extracting archive
0/107 [=====] 0%
 9/107 [==>-----] 8%
11/107 [==>-----] 10%
18/107 [====>----] 16%
25/107 [=====>---] 23%
31/107 [=====>---] 28%
35/107 [=====>---] 32%
45/107 [=====>---] 42%
51/107 [=====>---] 47%
55/107 [=====>---] 51%
60/107 [=====>---] 56%
65/107 [=====>---] 60%
72/107 [=====>---] 67%
75/107 [=====>---] 70%
83/107 [=====>---] 77%
86/107 [=====>---] 80%
93/107 [=====>---] 86%
97/107 [=====>---] 90%
105/107 [=====>---] 98%
106/107 [=====>---] 99%
107/107 [=====>---] 100%
59 package suggestions were added by new dependencies, use `composer suggest` to see details.
Generating optimized autoload files
```

2. Set up the database connection in the `.env` file.

```
Which database will your application use? [SQLite]:
[sqlite ] SQLite
[mysql  ] MySQL
[mariadb] MariaDB
[pgsql  ] PostgreSQL (Missing PDO extension)
[sqsrsv ] SQL Server (Missing PDO extension)
> mysql
mysql

Default database updated. Would you like to run the default database migrations?
> yes

WARN The database 'book_management' does not exist on the 'mysql' database.
Would you like to create it? (yes/no) [yes]
>
INFO Preparing database.

Creating migration table .....

INFO Running migrations.

0001_01_01_000000_create_users_table .....
0001_01_01_000001_create_cache_table .....
0001_01_01_000002_create_jobs_table .....

INFO Application ready in [book-management]. You can start your application.
→ cd book-management
→ npm install && npm run build
|
```

Navigate to the project

```
C:\>cd book-management

C:\book-management>|
```

Create the Database and Model

1. Create a migration and model for the `Book` entity:

```
C:\book-management>php artisan make:model Book -m

C:\book-management>php artisan make:model Book -m
[INFO] Model [C:\book-management\app\Models\Book.php] created successfully.
[INFO] Migration [C:\book-management\database\Migrations\2024_12_03_160822_create_books_table.php] created successfully.

C:\book-management>
```

Open VS Code

```
C:\book-management>code .

File Edit Selection View Go Run Terminal Help

EXPLORER
BOOK-MANAGEMENT
  app
  bootstrap
  config
  database
    factories
    migrations
      0001_01_01_000000_create_users_table.php
      0001_01_01_000001_create_cache_table.php
      0001_01_01_000002_create_jobs_table.php
      2024_12_03_160822_create_books_table.php
    seeders
```

2. Define the database schema in the migration file located in `database/migrations/`:
Look for the migration file

```
public function up(): void
{
    Schema::create('books', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->string('author');
        $table->text('description');
        $table->integer('published_year');
        $table->string('genre');
        $table->timestamps();
    });
}
```

3. Run the migration:

```
C:\book-management>code .

C:\book-management>php artisan migrate

C:\book-management>php artisan migrate
[INFO] Running migrations.
2024_12_03_160822_create_books_table ..... 27.41ms DONE

C:\book-management>
```

Create the Controller and Routes

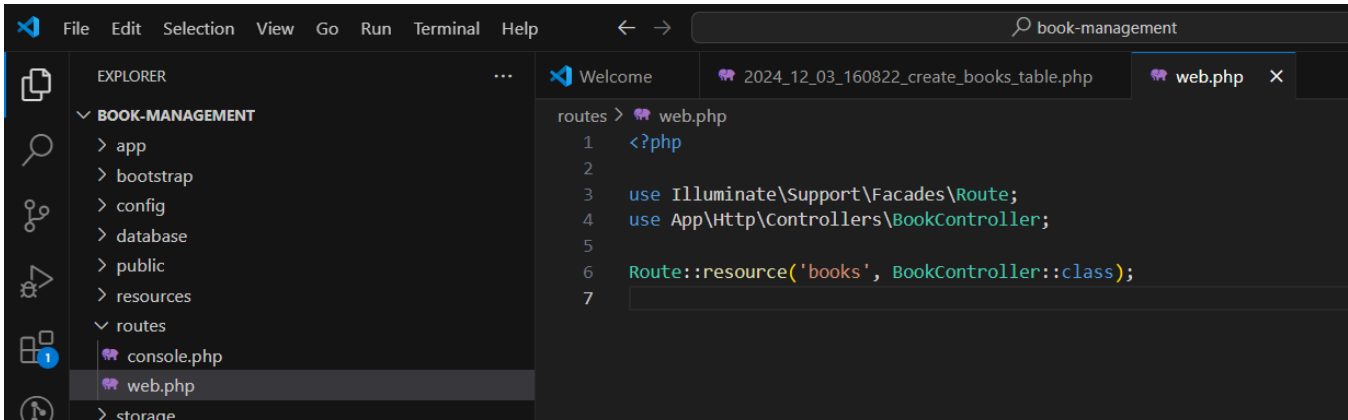
1. Generate a controller for books:

```
C:\book-management>php artisan make:controller BookController --resource

[INFO] Controller [C:\book-management\app\Http\Controllers\BookController.php] created successfully.

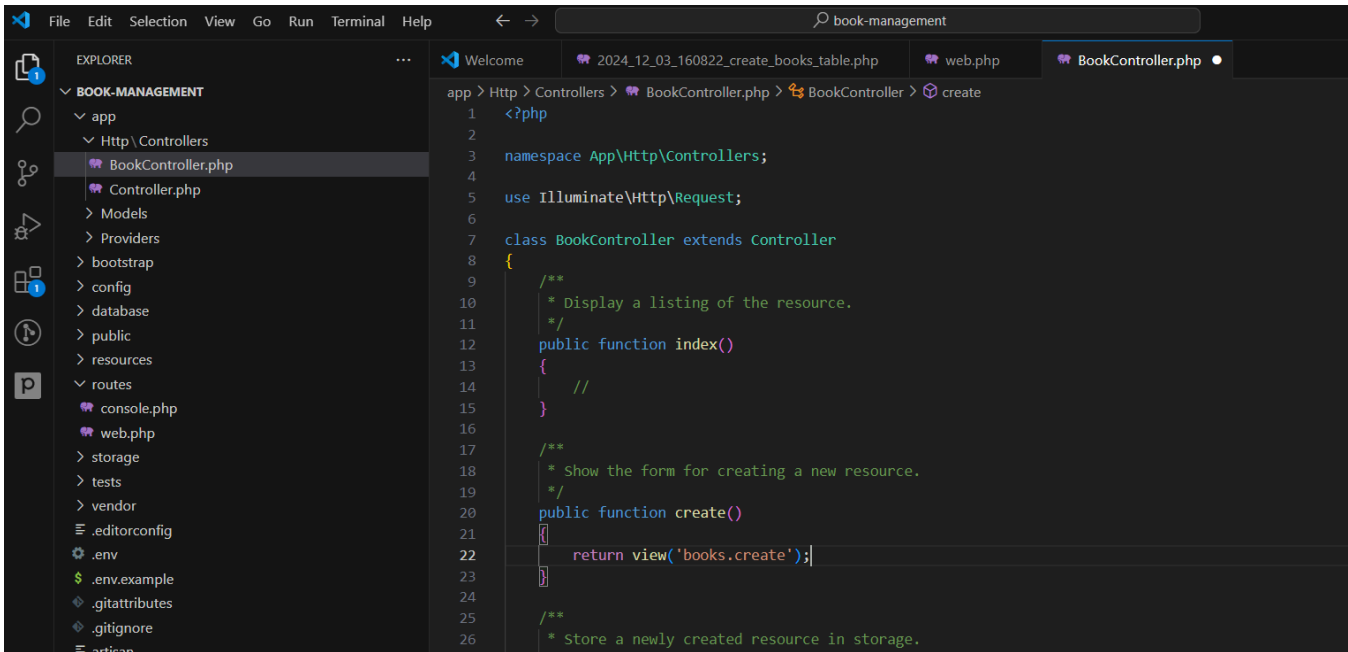
C:\book-management>
```

2. Define the routes in `routes/web.php`:

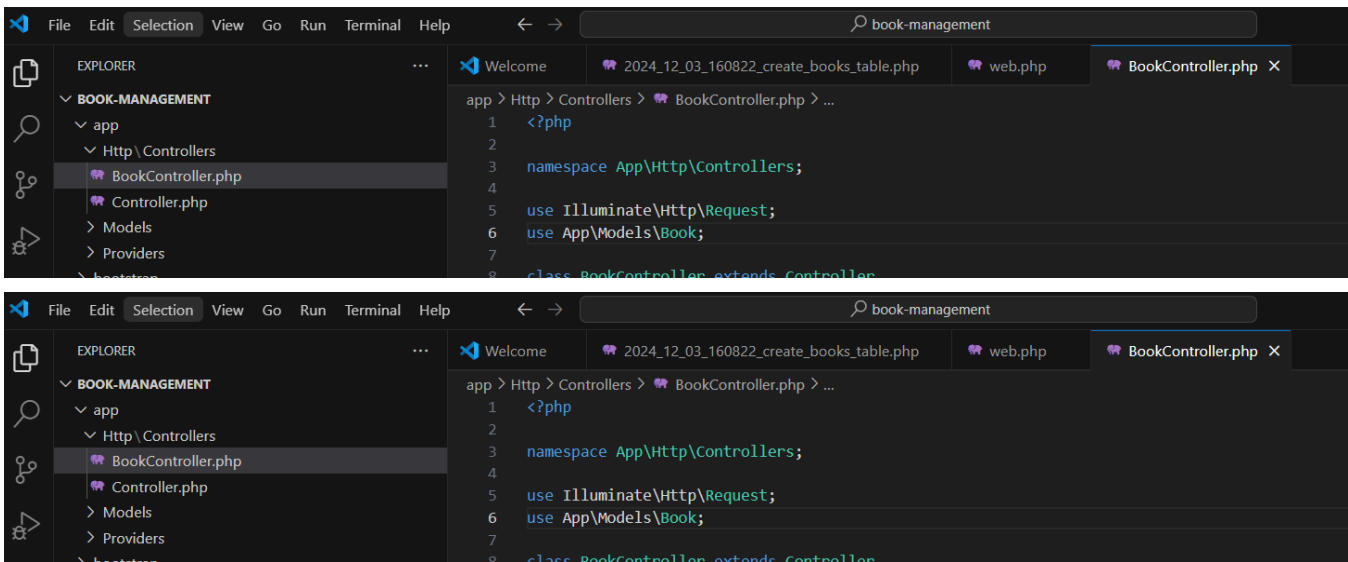


Implement CRUD Functionalities
a. Create and Store Books

1. In the `create` method of `BookController`, return a view for adding a new book:



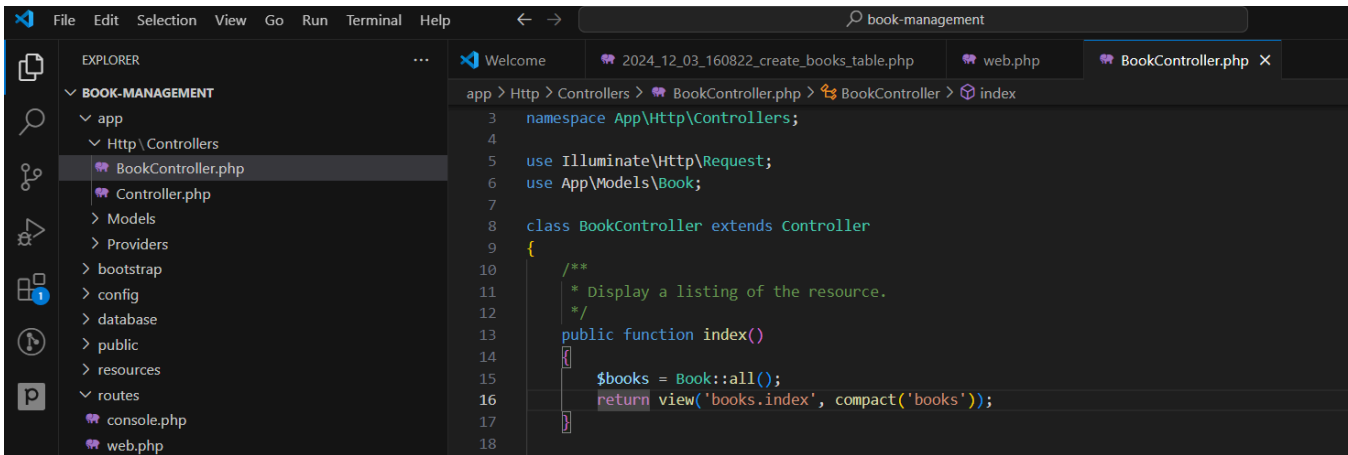
2. In the `store` method, validate and save the book data:
Import Book model



3. Create the view `resources/views/books/create.blade.php` with a form to add a book.

b. Read and List Books

1. In the `index` method of `BookController`, retrieve all books and return a view:



```
File Edit Selection View Go Run Terminal Help
book-management

EXPLORER
BOOK-MANAGEMENT
  app
    Http\Controllers
      BookController.php
      Controller.php
    Models
    Providers
    bootstrap
    config
    database
    public
    resources
    routes
    console.php
    web.php

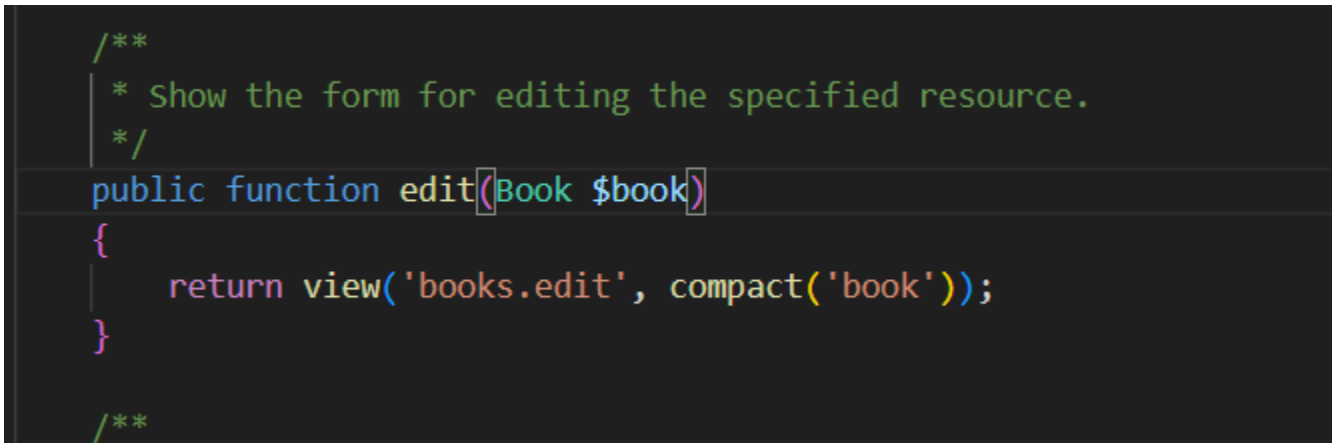
Welcome
2024_12_03_160822_create_books_table.php
web.php
BookController.php X

app > Http > Controllers > BookController.php > BookController > index
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\Book;
7
8 class BookController extends Controller
9 {
10     /**
11      * Display a listing of the resource.
12      */
13     public function index()
14     {
15         $books = Book::all();
16         return view('books.index', compact('books'));
17     }
18 }
```

2. Create the view `resources/views/books/index.blade.php` to display the list of books.

c. Edit and Update Books

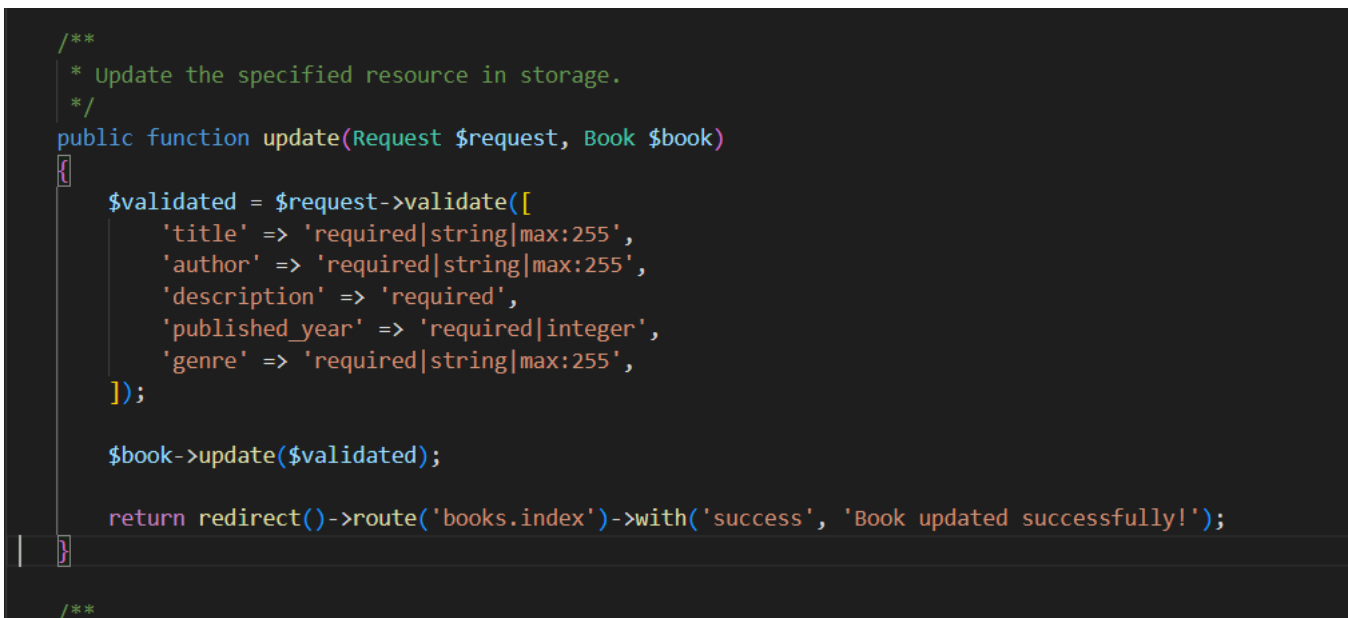
1. In the `edit` method, retrieve the book data and return a view:



```
/**
 * Show the form for editing the specified resource.
 */
public function edit(Book $book)
{
    return view('books.edit', compact('book'));
}

/**
```

2. In the `update` method, validate and update the book data:



```
/**
 * Update the specified resource in storage.
 */
public function update(Request $request, Book $book)
{
    $validated = $request->validate([
        'title' => 'required|string|max:255',
        'author' => 'required|string|max:255',
        'description' => 'required',
        'published_year' => 'required|integer',
        'genre' => 'required|string|max:255',
    ]);

    $book->update($validated);

    return redirect()->route('books.index')->with('success', 'Book updated successfully!');
}

/**
```

3. Create the view `resources/views/books/edit.blade.php` with a form to edit a book.

d. Delete Books

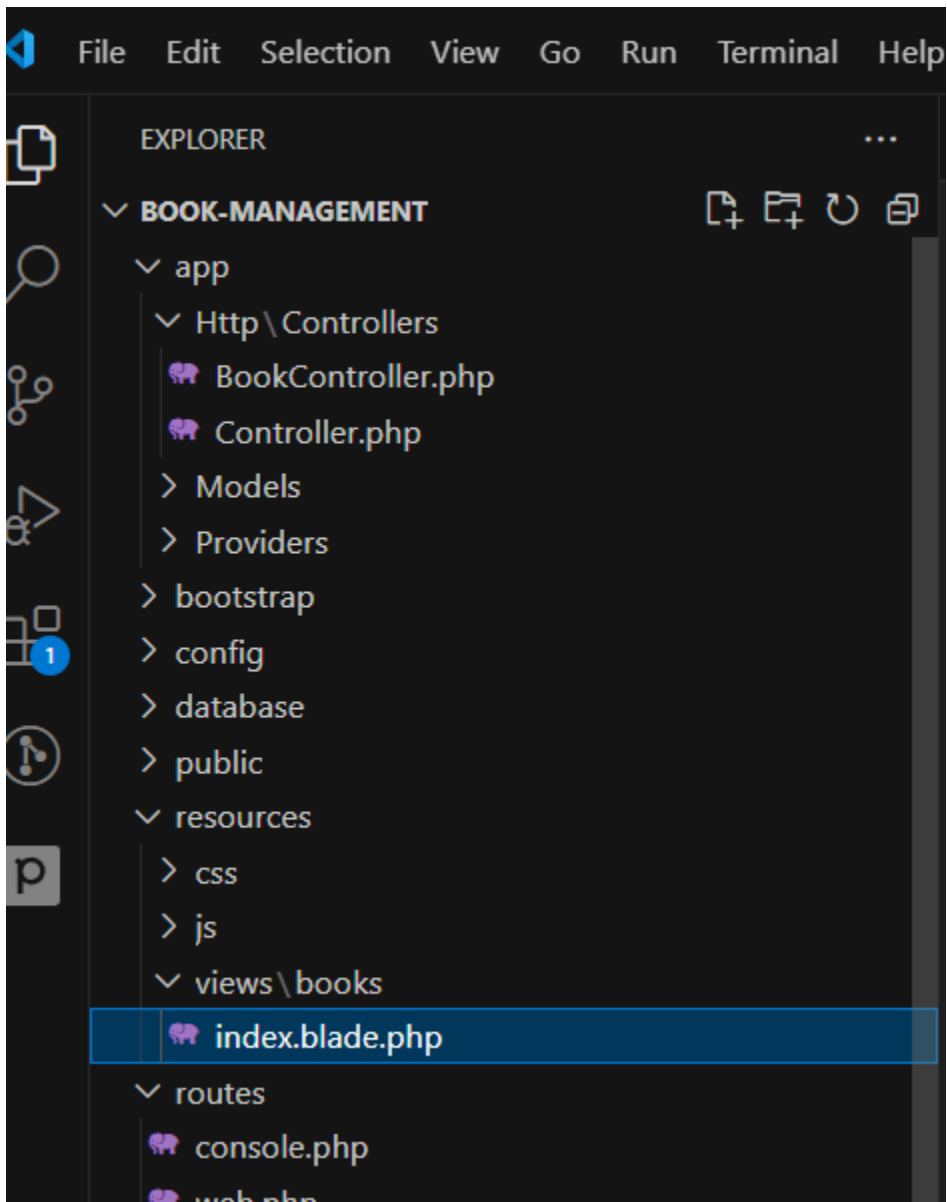
1. In the `destroy` method, delete the selected book:

```
/**
 * Remove the specified resource from storage.
 */
public function destroy(Book $book)
{
    $book->delete();
    return redirect()->route('books.index')->with('success', 'Book deleted successfully!');
}
```

2. Add delete buttons in the `index` view for each book.

Implement Search Functionality

1. Add a search form in the `index` view:



Copy the HTML code from the repository

Copy and paste this `index.blade.php` and `edit.blade.php` to `resources/views/books`, <https://github.com/jayimillena/book-management/tree/main/resources/views/books>

2. Update the `index` method to handle search queries:

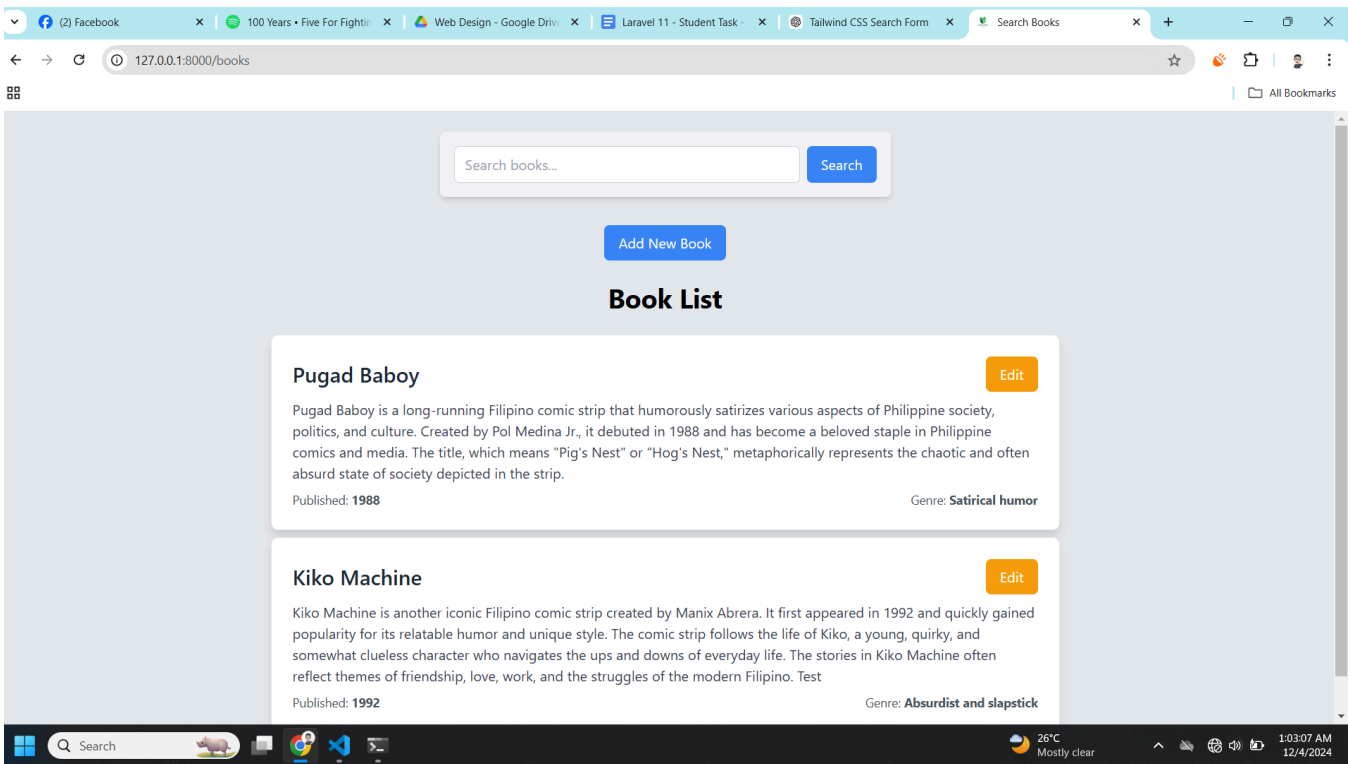
```
app > Http > Controllers > BookController.php > BookController > index
1 use App\Models\Book;
2
3
4
5
6
7
8 class BookController extends Controller
9 {
10     /**
11      * Display a listing of the resource.
12      */
13     public function index(Request $request)
14     {
15         $query = $request->input('query');
16
17         $books = Book::when($query, function ($q) use ($query) {
18             return $q->where('title', 'like', "%$query%")
19                 ->orWhere('author', 'like', "%$query%");
20         })->get();
21
22         return view('books.index', compact('books'));
23     }
24
25     /**
26      * Show the form for creating a new resource.
```

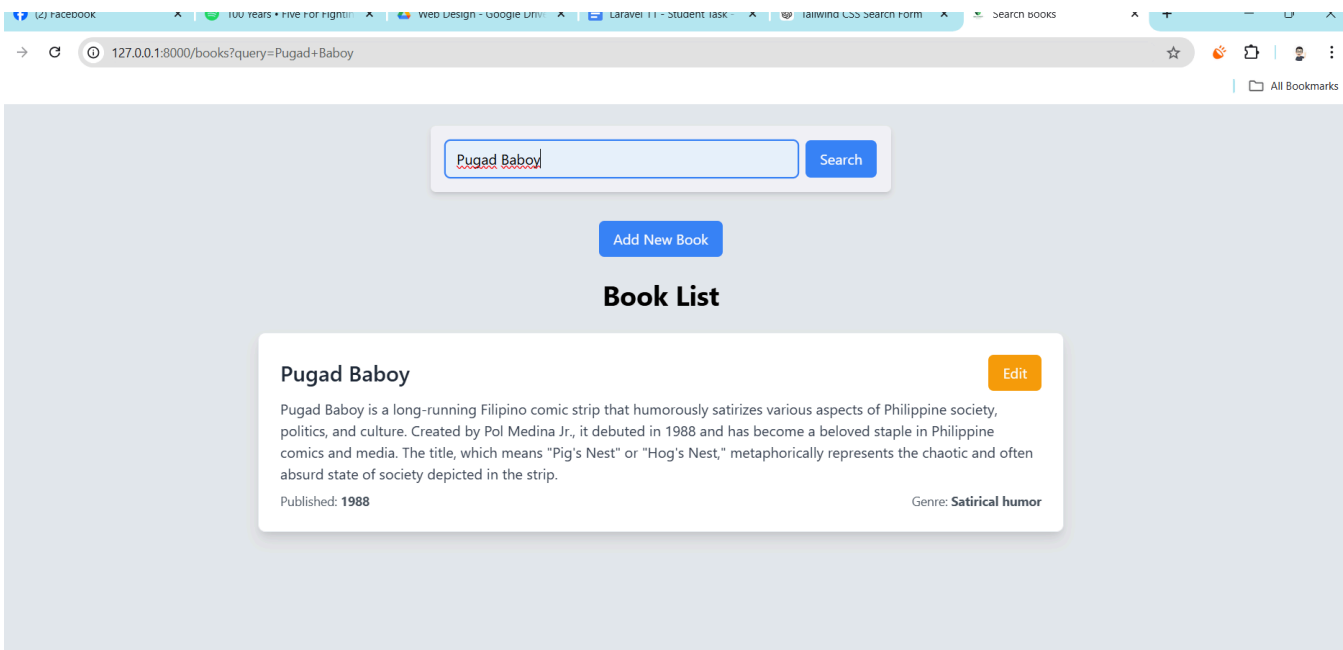
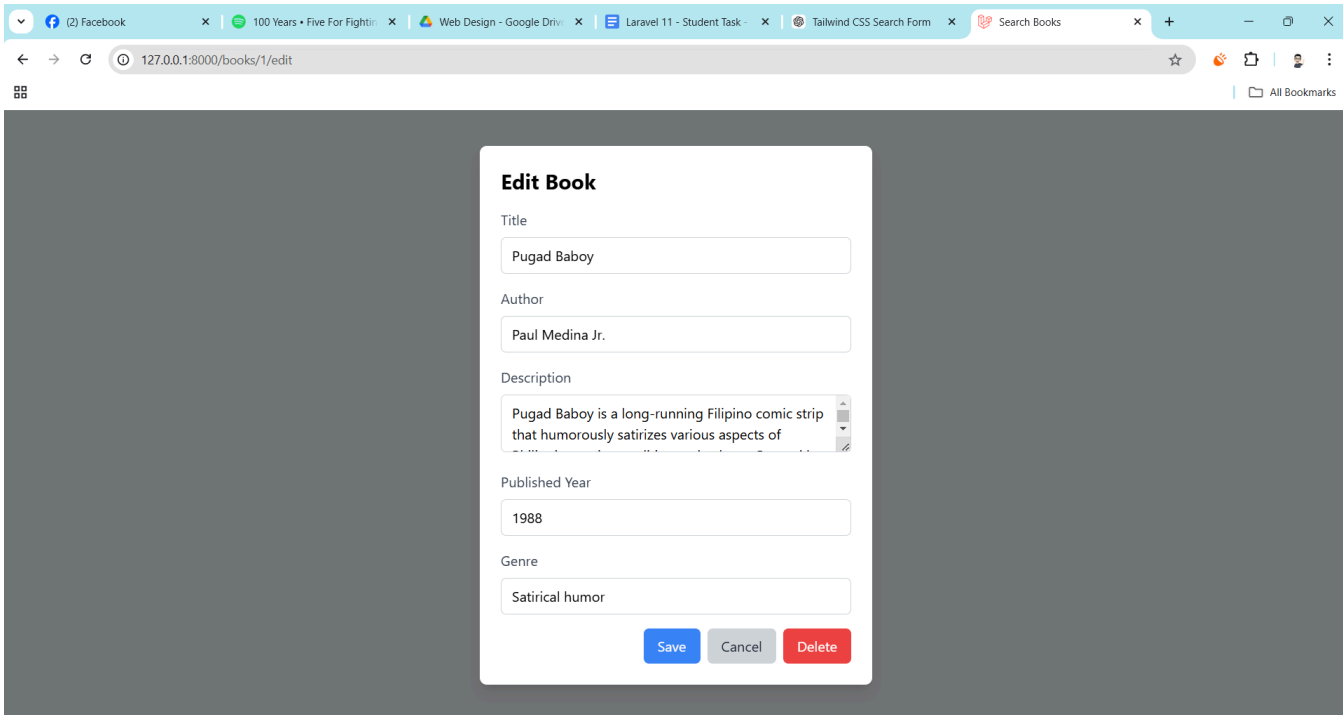
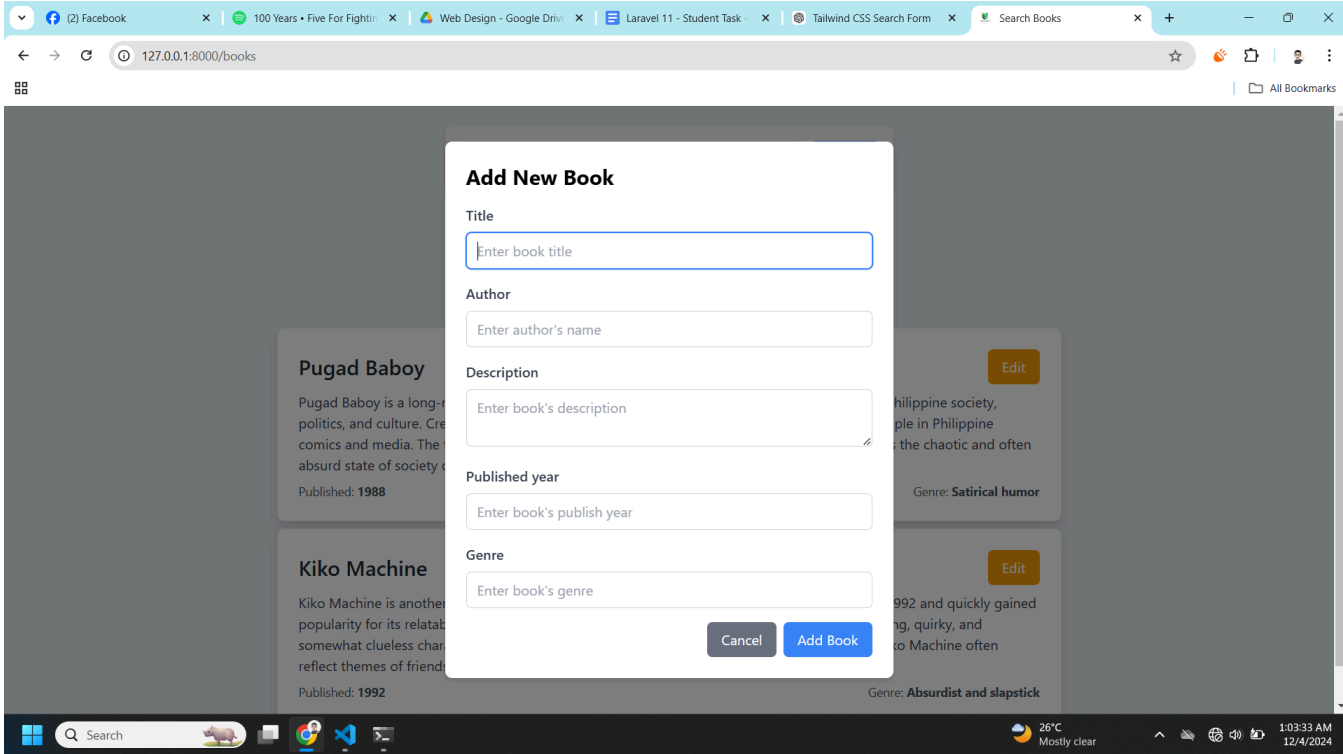
Update the web.php

```
routes > web.php
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\BookController;
5
6 Route::get('/', [BookController::class, 'index'])->name('books.index');
7 Route::resource('books', BookController::class);
8
```

Test Your Application

- Add sample data to the database.
- Verify that all functionalities (search, create, edit, update, delete) work as expected.





Deliverables

- 1. A GitHub repository containing the Laravel project.
- 2. A short document explaining the functionality and how to run the project.

Criteria for Grading

Here's a rubric template for evaluating web development projects using Laravel and Tailwind CSS. This rubric divides evaluation into three categories with detailed criteria:

Web Development Rubric for Laravel & Tailwind CSS Projects

1. Functionality (40 points)

- **Excellent (36-40 points):** The application fully meets all functional requirements. All features, such as CRUD operations, form validation, authentication, and other Laravel functionalities, work seamlessly without any bugs.
- **Good (28-35 points):** The application meets most functional requirements, with only minor issues or incomplete features that do not significantly impact the user experience.
- **Satisfactory (20-27 points):** The application meets some functional requirements, but several features have issues or are incomplete. This affects usability.
- **Needs Improvement (0-19 points):** The application fails to meet key functional requirements, with major bugs or incomplete core features that impact overall usability.

2. Design and UI/UX (30 points)

- **Excellent (27-30 points):** The application has an appealing, responsive, and intuitive design using Tailwind CSS. The layout is well-structured, and elements are styled consistently. Accessibility considerations are taken into account.
- **Good (21-26 points):** The design is visually appealing and mostly responsive, but there are minor inconsistencies in styling or a few usability issues.
- **Satisfactory (15-20 points):** The design is functional but may lack polish or have issues with responsiveness or accessibility. Styling inconsistencies are present.
- **Needs Improvement (0-14 points):** The design is not user-friendly or responsive, and styling is inconsistent or not aligned with best practices. Accessibility considerations are lacking.

3. Code Quality and Best Practices (30 points)

- **Excellent (27-30 points):** Code is clean, well-organized, and adheres to Laravel and Tailwind CSS best practices. Proper naming conventions, modular structure, and efficient use of resources (e.g., blade components and Tailwind utility classes) are evident. Documentation is clear.
- **Good (21-26 points):** Code is mostly well-organized and follows best practices, with minor deviations. Most components are modular, and documentation is adequate but could be improved.
- **Satisfactory (15-20 points):** Code structure is somewhat organized but contains noticeable issues like code duplication or improper use of Laravel and Tailwind CSS conventions. Documentation is sparse or unclear.
- **Needs Improvement (0-14 points):** Code is disorganized, difficult to read, or violates core best practices for Laravel and Tailwind CSS. Significant issues with code modularity or documentation.

Total: 100 points

This rubric can be adjusted as needed based on project specifications or additional criteria.

Instructor A Deliverables

Save you file as ``[Laravel Student Task]-[students last name]-[student_first name]-[date today].pdf``
Example: `Laravel Student Task-Millena-Jay-December 9, 2024.pdf`

Deadline of Submission

The deadline is set for December 5, 2024. Please send it to me via private message on my Messenger at [www.fb.com/knightofdaraga](https://www.facebook.com/knightofdaraga) | Jay Millena - Instructor A

**Reference:**

Netflix. (2024). *Einstein and the bomb: Release date, trailer, and news*. Netflix Tudum. Retrieved December 3, 2024, from

<https://www.netflix.com/tudum/articles/einstein-and-the-bomb-release-date-trailer-news>

Netflix. (2024). Einstein and the Bomb [Docudrama]. Netflix. Retrieved from <https://www.netflix.com>