Домашняя работа
по предмету:
Продвинутое программирование на PHP — Laravel

Выполнил: Байбородин Александр

Урок 6. Работа с формами

Задание
Цели практической работы:

Научиться:

— связывать данные модели с полями формы;
— обеспечивать безопасность формы с помощью CSRF-токенов;
— валидировать поля формы;
— использовать семантически правильные и релевантные элементы управления формой.

Что нужно сделать:

В этой практической работе вы создадите форму по добавлению новой книги в книжный каталог. Форма будет создаваться внутри шаблона. Данные из формы будут записываться в соответствующие поля базы данных с помощью модели Eloquent.

1. Внутри директории resources/view корневого каталога проекта создайте новый блейд-шаблон с именем form.blade.php.
Пример формы:

book-add-form.blade.php

```
<div class="add-books__form-wrapper">
      <form name="add-new-book" id="add-new-book" method="post" action="
({url('store-form')})">
        <div class="form-section">
          <label for="title">Title</label>
          <input type="text" id="title" name="title" class="form-control"
required>
        </div>
        <div class="form-section">
          <label for="author">Author</label>
         <input type="text" id="author" name="author" class="form-control"
required>
        </div>
         <div class="form-section">
          <label for="genre">Choose Genre:</label>
           <select name="genre" id="genre">
             <option value="fantasy">Fantasy</option>
             <option value="sci-fi">Sci-Fi</option>
             <option value="mystery">Mystery</option>
             <option value="drama">Drama</option>
           </select>
        </div>
        <button type="submit" class="btn btn-primary">Submit</button>
      </form>
</div>
```

В примере выше продемонстрирована простая форма для добавления новой записи о книге. В ней указаны поля с названием книги, именем автора, а также жанр, который можно выбрать из списка. Вы также можете добавить произвольные поля, чтобы сделать данные из формы более комплексными и приближенными к реальности.

2. Чтобы защитить данные формы от межсайтовой подделки запроса, добавьте внутрь формы CSRF токен. Напомним, сделать этом можно с помощью директивы @csrf или скрытого поля input:

csrf-token

```
<form method="POST" action="/profile">
    @csrf

    <!-- Или... -->
    <input type="hidden" name="_token" value="{{ csrf_token() }}" />
</form>
```

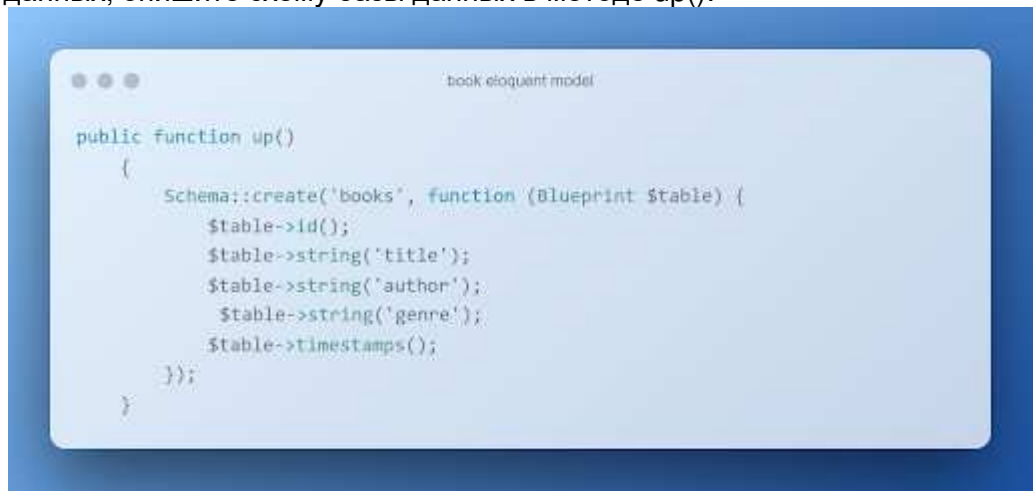
3. Свяжите данные полей формы с моделью Laravel. Для этого создайте новую модель. Сделать это можно из командной строки с помощью artisan-команды:

make model

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

Напомним, что флаг -**mfsc** создаст модель, наполнитель, контроллер и файл миграции.

4. Чтобы данные из формы корректно записывались в соответствующие поля базы данных, опишите схему базы данных в методе up():

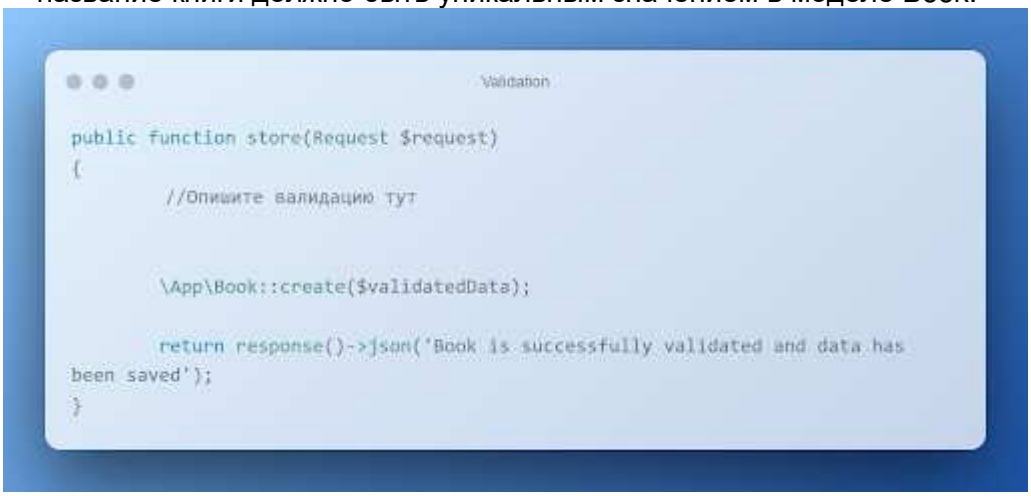


Чтобы в базе данных появились соответствующие поля, не забудьте повторно запустить миграции в базе данных, воспользовавшись соответствующей командой artisan.

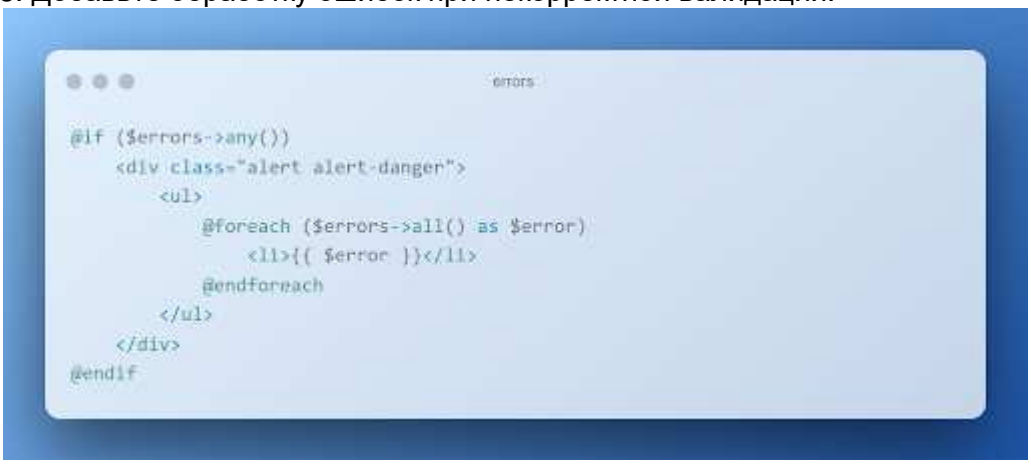
5. Внутри файла /routes/web.php опишите новый роут (метод GET), который будет вызывать метод **index** контроллера **BookController** по url /**index**. Также добавьте роут с методом POST, который будет вызывать метод store того же контроллера **BookController** с url** /store**

6. Опишите метод index внутри контроллера BookController. Метод должен возвращать представление формы в браузере.

7. Опишите метод store(). Прежде чем сохранить данные внутри модели, проведите валидацию с помощью метода $request->validate(). Правила для валидации:
— все поля обязательны к заполнению, без пустых строк и пробелов в качестве единственного значения;
— максимальное число символов в имени автора — 100, в названии книги — 255;
— название книги должно быть уникальным значением в модели Book.



8. Добавьте обработку ошибок при некорректной валидации.



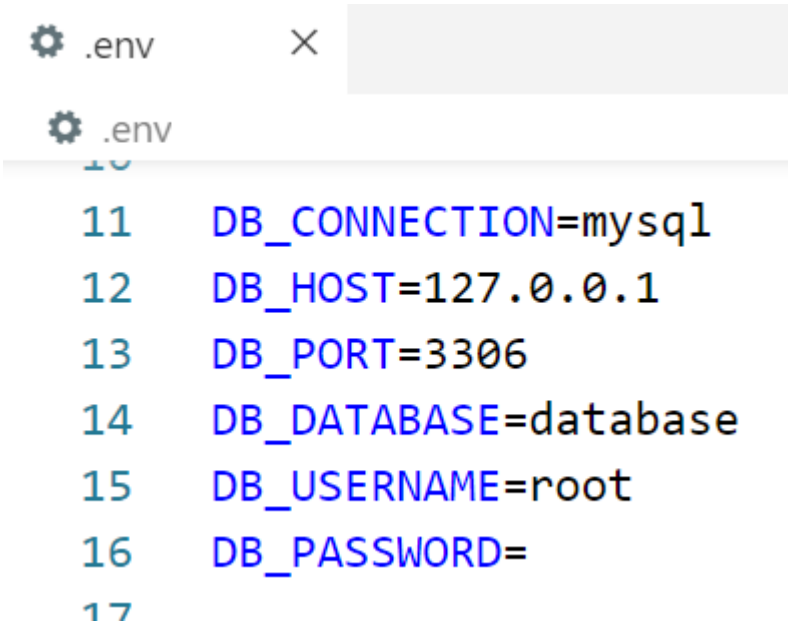
Результат

Предварительно в начале создадим проект laravel:

composer create-project laravel/laravel my-laravel-app-lesson-six

```
  - Installing myclabs/deep-copy (1.13.0): Extracting archive
  - Installing phpunit/phpunit (10.5.45): Extracting archive
  - Installing spatie/error-solutions (1.1.3): Extracting archive
  - Installing spatie/backtrace (1.7.1): Extracting archive
  - Installing spatie/flare-client-php (1.10.1): Extracting archive
  - Installing spatie/ignition (1.15.0): Extracting archive
  - Installing spatie/laravel-ignition (2.9.0): Extracting archive
64 package suggestions were added by new dependencies, use `composer suggest` to
 see details.
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

  INFO  Discovering packages.

  laravel/sail ..................................................................
..... DONE
  laravel/sanctum ...............................................................
..... DONE
  laravel/tinker ................................................................
..... DONE
  nesbot/carbon .................................................................
..... DONE
  nunomaduro/collision ..........................................................
..... DONE
  nunomaduro/termwind ...........................................................
..... DONE
  spatie/laravel-ignition .......................................................
..... DONE

82 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

  INFO  No publishable resources for tag [laravel-assets].

No security vulnerability advisories found.
> @php artisan key:generate --ansi

  INFO  Application key set successfully.


A@LAPTOP-RHI13GV6 MINGW64 /c/laravelapp
$
```

Предварительно настроем также соединение с базой database MySQL в файле .env:



Теперь по пунктам выполним пункты задания.

1. Создание шаблона формы

Создадим файл form.blade.php в директории resources/views:

```
<div class="add-books_form-wrapper">
   <form name="add-new-book" id="add-new-book" method="post" action="{{ url('/store') }}">
      @csrf
      <div class="form-section">
         <label for="title">Title</label>
         <input type="text" id="title" name="title" class="form-control" required>
      </div>
      <div class="form-section">
         <label for="author">Author</label>
         <input type="text" id="author" name="author" class="form-control" required>
      </div>
      <div class="form-section">
         <label for="genre">Choose genre:</label>
         <select name="genre" id="genre">
            <option value="fantasy">Fantasy</option>
            <option value="sci-fi">Sci-Fi</option>
            <option value="mystery">Mystery</option>
            <option value="drama">Drama</option>
         </select>
      </div>
      <button type="submit" class="btn btn-primary">Submit</button>
   </form>
</div>
```

2. Добавление CSRF токена

CSRF токен уже добавлен в форму с помощью директивы @csrf.

3. Создание модели и миграции

Создадим модель, миграцию, фабрику и контроллер с помощью команды Artisan:

```
$ php artisan make:model Book -mfsc

  INFO  Model [C:\laravelapp\my-laravel-app-lesson-six\app\Models\Book.php] created successfully.

  INFO  Factory [C:\laravelapp\my-laravel-app-lesson-six\database\factories\BookFactory.php] created successfully.

  INFO  Migration [C:\laravelapp\my-laravel-app-lesson-six\database\migrations/2025_02_16_182128_create_books_table.php] created successfully.

  INFO  Seeder [C:\laravelapp\my-laravel-app-lesson-six\database\seeders\BookSeeder.php] created successfully.

  INFO  Controller [C:\laravelapp\my-laravel-app-lesson-six\app\Http\Controllers\BookController.php] created successfully.
```

Поправим код модели app/Models/Book.php:

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Book extends Model
{
    use HasFactory;

    protected $fillable = [
        'title',
        'author',
        'genre',
    ];
}
```

4. Описание схемы базы данных

Откроем созданный файл миграции 2025_02_16_182128_create_books_table.php и опишем схему базы данных в методе up():

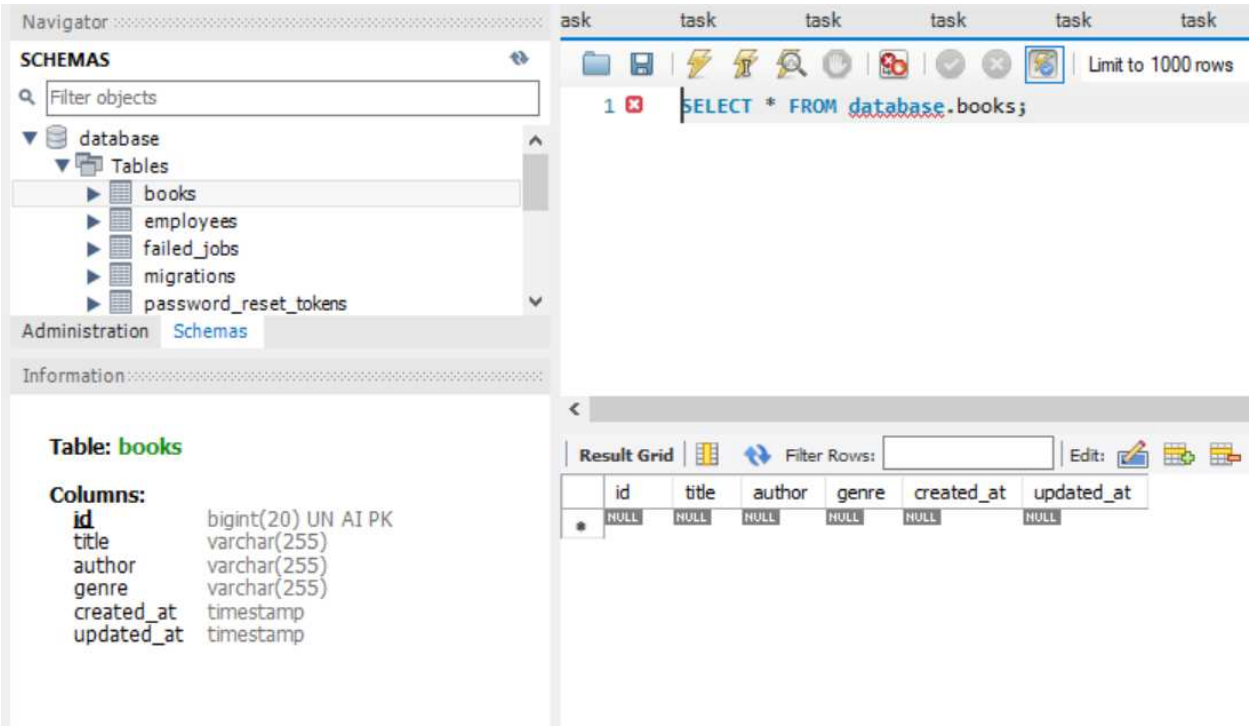
```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up()
    {
        Schema::create('books', function (Blueprint $table) {
            $table->id();
            $table->string('title');
            $table->string('author');
            $table->string('genre');
            $table->timestamps();
        });
    }
```

```php
    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('books');
    }
};
```

Запустим миграцию:

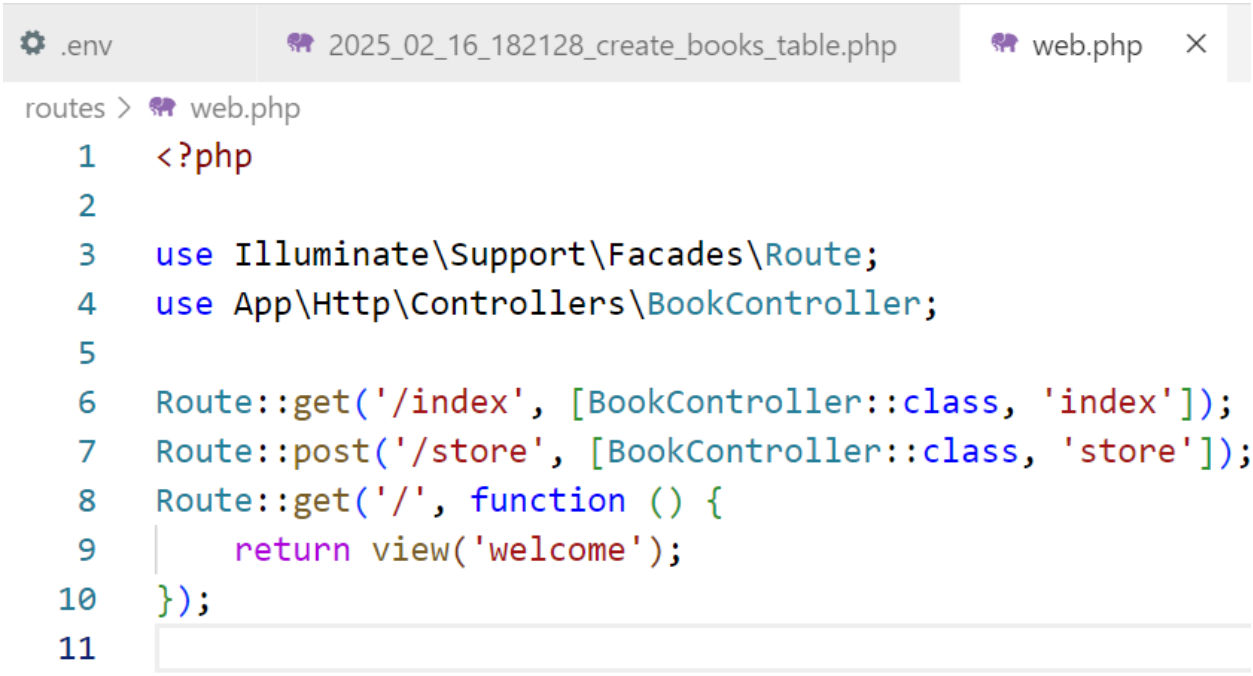
```
$ php artisan migrate

  INFO  Running migrations.

  2025_02_16_182128_create_books_table ..... 16ms DONE

A@LAPTOP-RHI13GV6 MINGW64 /c/laravelapp/my-laravel-app-lesson-six
$
```

Видим, что таблица создалась:



5. Создание роутов

Добавим роуты в файл routes/web.php:



6. Описание метода index в контроллере

Откроем файл app/Http/Controllers/BookController.php и опишим метод index:

BookController.php

app > Http > Controllers > BookController.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class BookController extends Controller
{
    public function index()
        {
            return view('form');
        }
}

```

7. Описание метода store и валидация

Опишем метод store в контроллере и добавим валидацию:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class BookController extends Controller
{
    public function index()
    {
        return view('form');
    }

    public function store(Request $request)
    {
        $validatedData = $request->validate([
            'title' => 'required|unique:books|max:255',
            'author' => 'required|max:100',
```

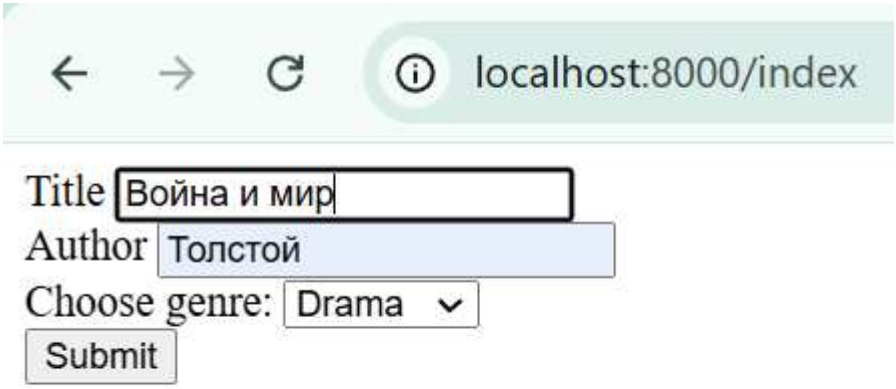
```
            'genre' => 'required',
        ]);

        \App\Models\Book::create($validatedData);

        return response()->json('Book is successfully validated and data has
been saved');
    }
}
```

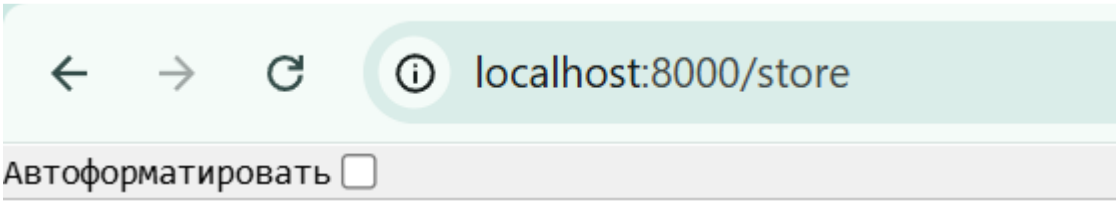
8. Обработка ошибок валидации

Добавим обработку ошибок в шаблоне формы:

```
<div class="add-books_form-wrapper">
    <form name="add-new-book" id="add-new-book" method="post" action="{{
url('/store') }}">
        @csrf
        @if ($errors->any())
        <div class="alert alert-danger">
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
        @endif
        <div class="form-section">
            <label for="title">Title</label>
            <input type="text" id="title" name="title" class="form-control" required>
        </div>
        <div class="form-section">
            <label for="author">Author</label>
            <input type="text" id="author" name="author" class="form-control"
required>
        </div>
        <div class="form-section">
            <label for="genre">Choose genre:</label>
            <select name="genre" id="genre">
                <option value="fantasy">Fantasy</option>
                <option value="sci-fi">Sci-Fi</option>
                <option value="mystery">Mystery</option>
                <option value="drama">Drama</option>
            </select>
        </div>
        <button type="submit" class="btn btn-primary">Submit</button>
    </form>
</div>
```

Запустим сервер командой php artisan serve и успешно тестируем форму:





"Book is successfully validated and data has been saved"