Курсова работа по PHP  
**Тема:** *Система за книги*на  
**Мартин Тихомиров Тодоров**, ФН: **1909011223**,  
специалност - Софтуерно инженерство, **втори** курс

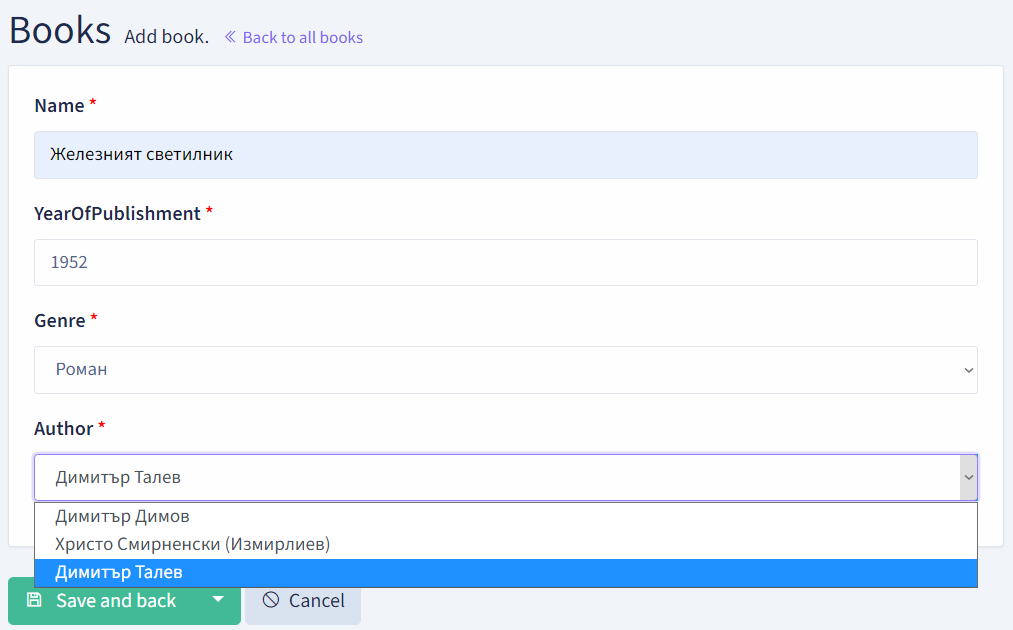
Документация на проекта ,,**PHPBooksProject**”  
Link към **GitHub** - https://github.com/MartinTT00/PHPBooksProject

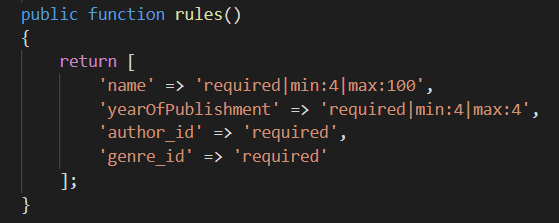
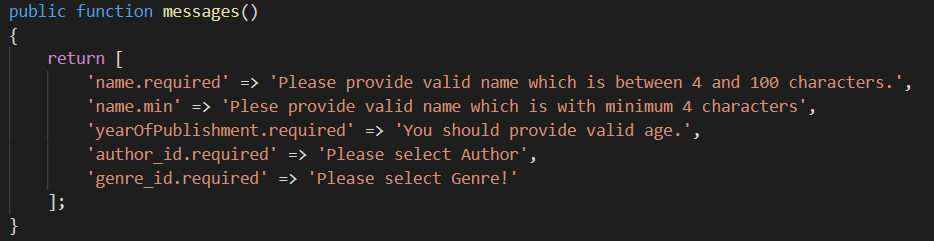
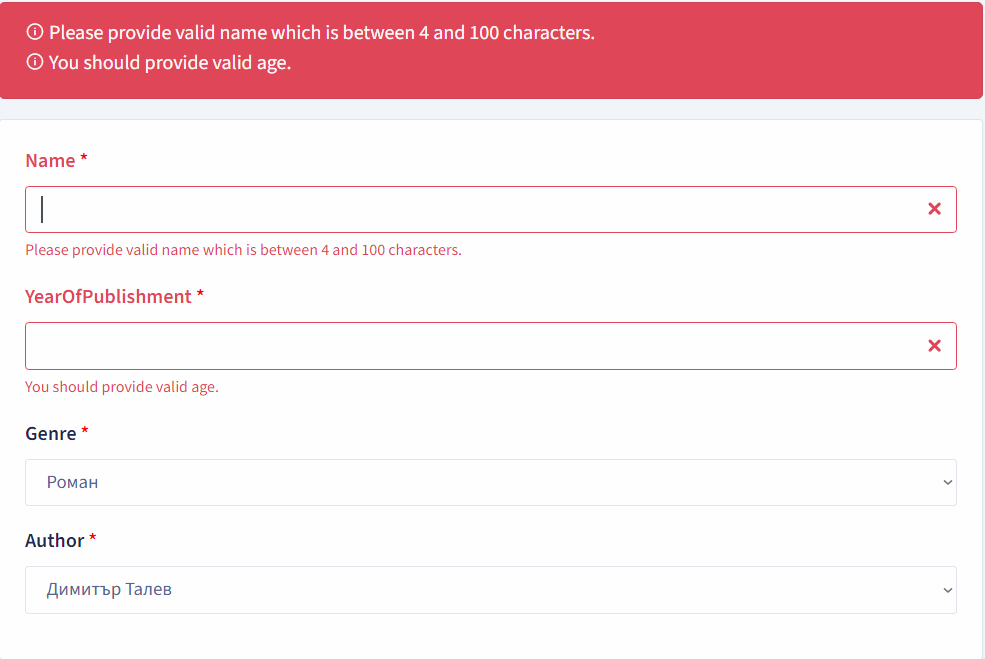
За старт на проекта, трябва да сме си конфигурирали Laravel, да сме си инсталирали Composer и едно от най-важните неща е да имаме XAMPP.

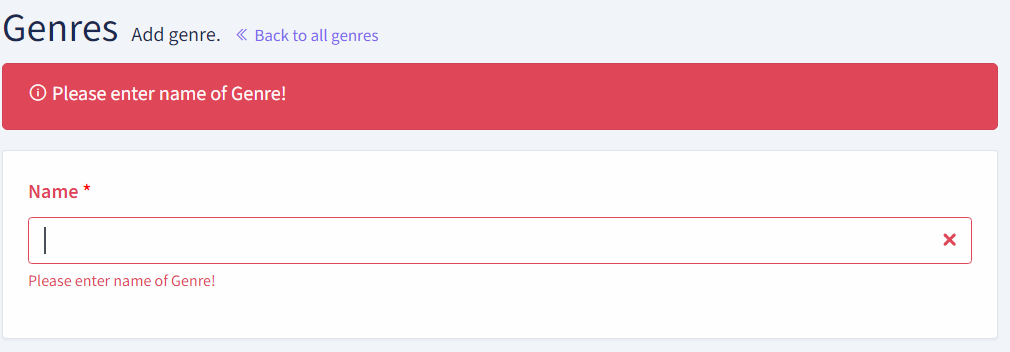
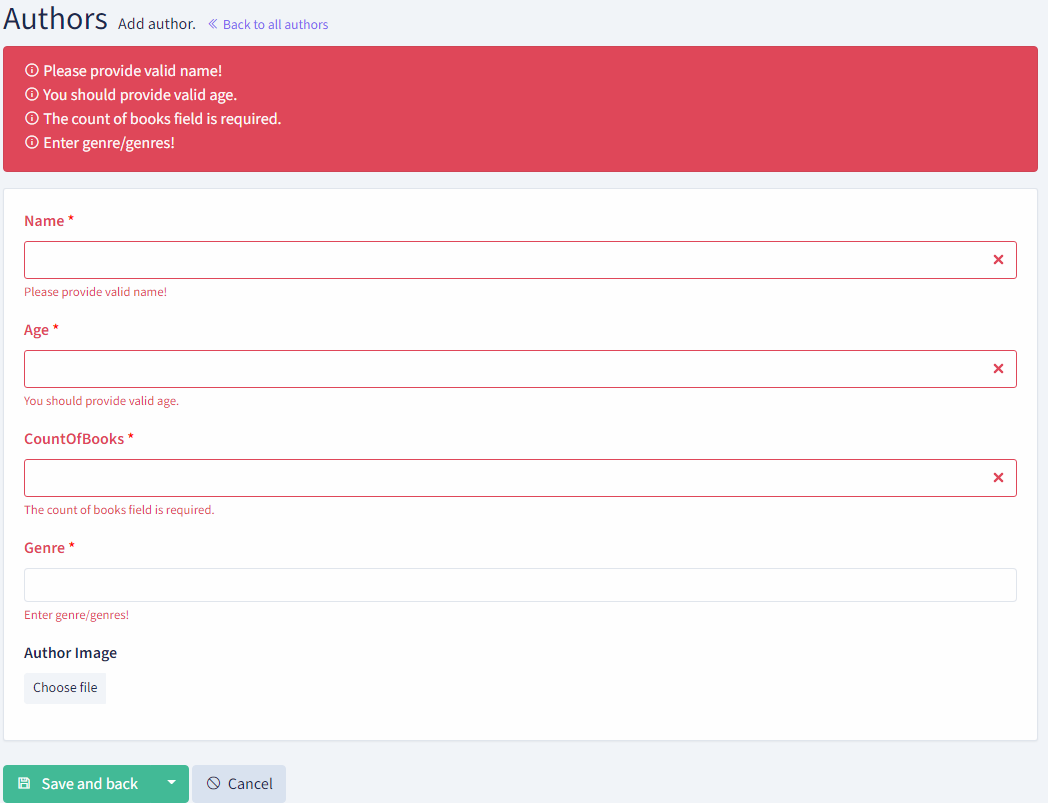
Проектът е направен на PHP Laravel и чрез допълнителната библиотека – Backpack, с чиято помощ са направени Admin panel-а, CRUD операциите за всеки от моделите, както и съответните им контролери и заявки.   
В cmd-то, таблиците на съответните модели се правят чрез:   
**php artisan make:migration create\_(име на таблицата)\_table**  
и ако правим някакви промени по миграцията или искаме тя да се запази и да се отрази в базата (phpMyAdmin), използваме командата:  
**php artisan migrate**  
Тази команда запазва всички промени направени досега по миграциите в кодовата част и чрез нея промените се отразяват и в базата ни.   
Всички промени и цялостното изграждане на проекта от началото до финализирането му, може да се проследи чрез commits от линка към github в заглавната част на документа.

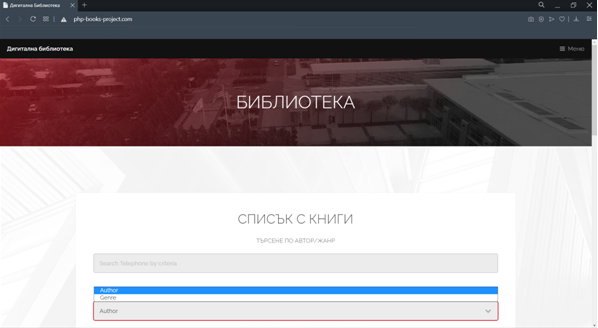
*Защо имаме админ панел и за какво служи той?*В админ панела имаме възможност за директно log-ване във формата, ако вече имаме регистрация, а ако нямаме, за да продължим трябва да си направим. След като се аутентикираме – системата знае кой сме и дали съществува такъв запис/админ. Админите могат да правят промени по моделите, да ги изтриват и създават, както и имат достъп до базата данни. Юзърът може само да чете и да търси в сайта, няма достъп до базата и не вижда и не може да прави промени върху, каквото и да е било по сайта.

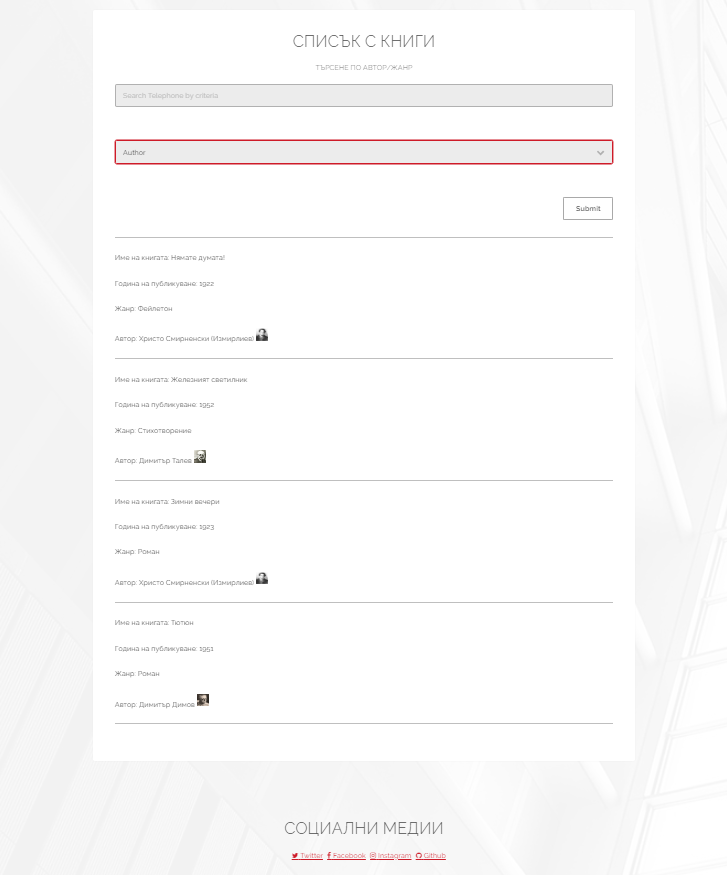
Основна структура на проекта, при изграждането му:  
  
Създаваме си три модела, които имат id (от тип integer) и timestamps, които се явяват 2 полета (Created\_at & Updated\_at) – тези полета са за всичките ни модели.  
  
Поотделно моделите ни имат и други полета, като:  
Моделът ни Жанр има поле от тип string – ‘name’, за име на жанра, което в същото време е уникално и не позволява да имаме два жанра с едно и също име.  
  
Моделът ни Автор има поле ‘name’ от тип string, което в този случай не е уникално и може да имаме двама автори с еднакво име. Този модел съдържа още 2 полета от тип integer 🡺 а именно ‘age’ и ‘countOfBooks’, които съответно представляват годините на автора и колко книги е написал той.  
  
Последният ни модел е Книга, която има също поле ‘name’ от тип string, което е уникално и има 3 полета от тип integer 🡺 ‘yearOfPublishment’, ‘author\_id’ и ‘genre\_id’, първото поле има за предназначение да показва през коя година е публикувана съответната книга, а останалите две са за да ги използваме за връзките между таблиците.  
  
Връзки между таблиците (Relationships)  
Моделите ни и съответно техните таблици са: Жанрове, Автори и Книги!  
Връзките между тях са:  
Един автор може да пише в много жанрове и да пише много книги.  
Една книга може да има един автор и един жанр.  
Един жанр може да има много книги и много автори, които пишат в този жанр.  
Връзката между автор и книга е 1-to-many  
Връзката между жанр и книга е 1-to-many  
Връзката между автор и жанр обаче е many-to-many, тук се включва и междинната ни таблица author\_genres, която има две полета от тип integer за двустранната ни връзка 1-to-many от страна на авторите и от страната на жанровете, тези полета са ‘author\_id’ & ‘genre\_id’   
Връзките са представени чрез дропдауни в контролера:   
Например в BookController имаме:  


Име на книгата  
Година през, която тя е публикувана  
Жанр, който избираме от дропдауна на вече съществуващи записи, които се взимат от “App\Models\Genre” , търси се по genre\_id от genre, но до юзъра не се връща id-то, а срещу съответното id, името (от ‘attribute’ => ‘name’)  
За избор на автор е по-аналогичен начин на жанра.  


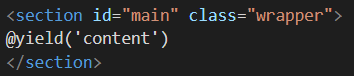
Админът може да извършва CRUD (Create, Read, Update, Delete) операции върху всичките модели, съответните им записи в таблиците и базата (Genres, Books, Authors)  
Като при авторите, при създаване или ъпдейтване (едитване) на някой обект може да се добави и снимка (без значение от формата .png или друг), след като се добави снимка, тя се запазва под .jpg разширение.   
public\storage\authors е директорията, в която се запазват снимките.  
Червените звезички от предната снимка до съответния label (етикет), показват, че полето има зададена валидация. За попълването на тези данни в съответното поле трябва да отговарят на зададени правила в Requests на съответния контролер, Например за BookRequest:  
  
Името на книгата – задължително (required) трябва да е попълнено, в противен случай ще се активизират съобщенията от функцията messages и няма да ни позволи да направим нов запис/промяна. Дължината на името на книгата трябва да е от минимум 4 символа до максимално 100 такива символа.  
За годината на публикуване е аналогично! Но при нея дължината на числото, което може да се въведе е от 4 до 4 символа, т.е. (0000-9999г), а полетата за автор и жанр, трябва да са избрани от дропдауна.  
  
  
  
  
Това ще ви излезе, ако се опитате да съзададете обект без да сте попълнили име на книгата и година на издаване, но сте попълнили автора и жанра.  


За останалите модели и технита таблици и полета е аналогично:  
  
   
Като при авторите могат да се избират повече от един жанр едновременно, не е задължително да се добавя снимка на автора, за това не е Required полето! Снимка може да се добави, ако сте админ и чрез ъпдейт може да добавите снимка или да промените съществуваща такава.  


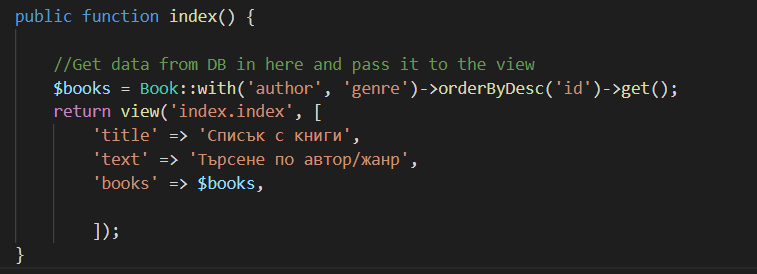
Страницата на сайта, която се вижда от потребителите е с интегриран темплейт, използвана е безплатна тема от <https://templated.co>   
  
Началната страница, която вижда юзъра е:  


  
  
Имаме листване на всички автори, тяхното сортиране и връщане в страницата им в обратен ред, на създаденото, колкото по скоро е създаден дадения обект, толкова по-нагоре в страницата ще излиза.  
В заданието имаме search-a да работи с два критерия (по автор и по жанр)  
Тук сме го реализирали чрез дропдаун, който проверява дали въведеният от нас текст е автор/жанр или нито едно от двете, ако е автор и съществува като запис, ни изпраща в друго вю – където ни дисплейва резултатите от търсенето, аналогично ако е жанр, малко по-късно ще представим функционалността, а ако не е нито едно от двете, тоест не съвпада търсенето, няма да имаме резултати!

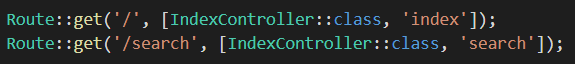
Когато интегрираме темплейта, html частта от главната страница си я копираме във view\layouts\  
app.blade.php и в секцията в която искаме да се дисплейват нашите резултати или избрана от нас логика във вюто, там пишем това:



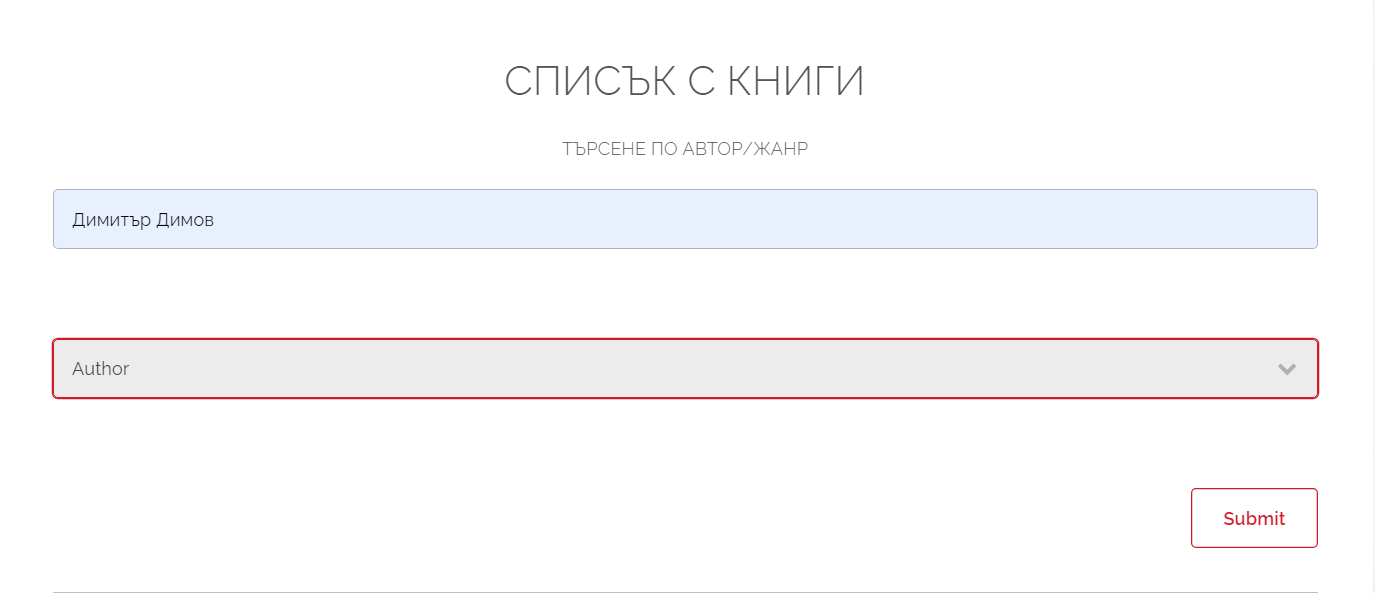
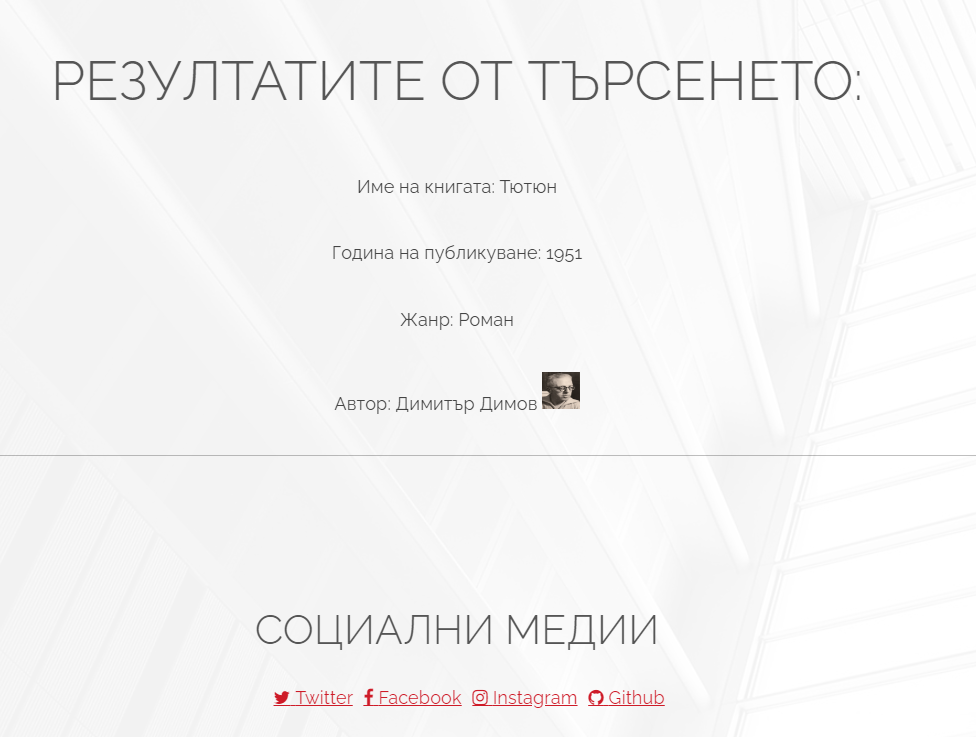
И за да използваме/интегрираме някаква логика си правим IndexController, който съдържа два метода (Index & Search)   
Index метода е отговорен за това листване на главната страница на записите, като те са сортирани и в обратен ред на създаването им.

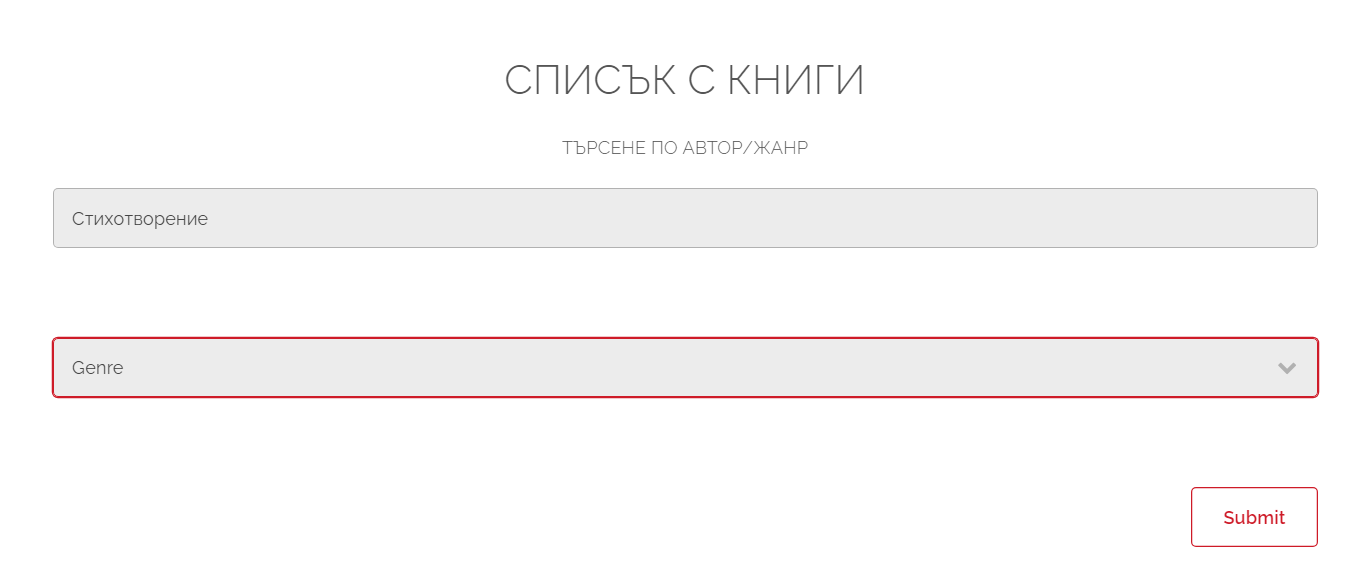


В другия метод (Search), търсим по 2 критерия, по жанр или по автор, ако въведеното е жанр, то във вюто (друго) ни връща всички книги в този жанр + техните връзки.  
В противен случай, ако въвденото е автор, то тогава взима всички книги написани от този автор и ни ги връща във вюто (друго). Ако въведеното не е нито автор, нито жанр, няма резултати от търсенето.  


Ако нямаме раутове за index и search, то те няма да работят 🡺 отиваме в routes\web.php и си ги добавяме:  


Вече видяхме как изглежда листването и началната страница, сега можем да видим и резултати от търсенето.

По критерий Автор!  
  


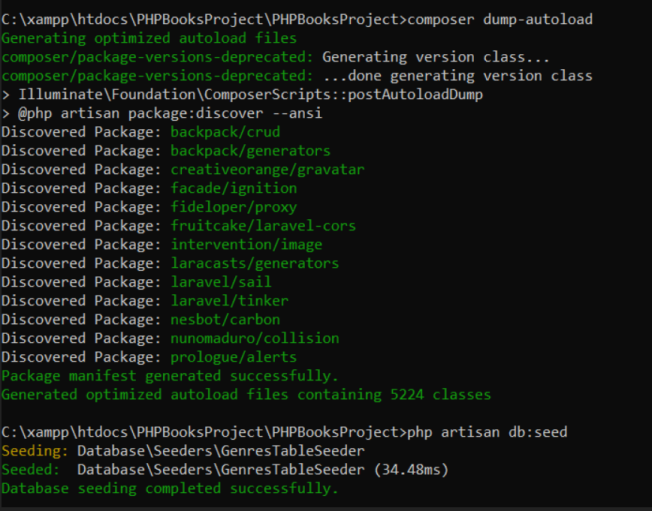
По критерий Жанр:  




Създаване на сийд   
От cmd-то влизаме в най-дъщерната директория на проекта чрез:  
cd.. C:\xampp\htdocs\PHPBooksProject\PHPBooksProject  
използваме командата ( ,,php artisan make:seeder GenresTableSeeder” )   
в този случай създаваме клас GenresTableSeeder.php , който се намира в database/seeders/GenresTableSeeder, добавяме си в use -   
,,use Illuminate\Support\Facades\DB”, за да можем да използваме класът DB  
DB::table(‘име на таблицата’)->insert([ [ данните за таблицата и полетата ]]);

В нашия случай:



След тази стъпка се връщаме в DatabaseSeeder.php и във функцията run(){}  
добавяме $this->call([GenresTableSeeder::class]) – извиква ни класа, който създадохме преди!  
Изтриваме си данните от базата данни, за да не се повтарят и в cmd-то използваме тази команда: ,,composer dump-autoload”, използваме я след всяка промяна по сийдовете, преди самото сийдване чрез:   
,,php artisan db:seed”  


За останалите таблици правим по аналогичен начин сийдовете!