**Система за управление на докторанти**

Системата е базирана на Laravel 8.x.

Започваме с инсталация на Xampp, който ще ни послужи за сетъп на virtual host за проекта. Редактираме файловете C:\xampp\apache\conf\extra\vhosts и C:\Windows\System32\drivers\etc\hosts, където ще създадем хоустовете. Инсталираме Composer. Той ни позволява да декларираме библиотеките свързани с нашия проект, да ги инсталира и ъпдейтва.

Следваща стъпка е инсталацията на самия Laravel. Това се случва посредством командния прозорец, където в директорията C:\xampp\htdocs\exercises подаваме командата “composer create-project –prefer-dist laravel/laravel име-на-папка” и задаваме директория, в която Composer инсталира пакетът от нужните файлове. Във .env файла трябва да редактираме url-а от localhost на име на подходящо за системата, като то трябва да съвпада с url-а във файловете vhosts и hosts(в тях посочваме и директорията на проекта).

Преминаваме към сетъп на admin panel. За целта в phpMyAdmin създаваме нова база данни – laravel. Admin panel ще създадем с помощта на Laravel Backpack, който е колекция от пакети за създаване на къстъм админ панел за сайтове и уеб апликейшъни. В командния прозорец задаме следните команди:

* composer require backpack/crud:"4.1.\*"
* composer require backpack/generators --dev
* composer require laracasts/generators --dev
* php artisan backpack:install

Всички пакети са инсталирани и базата данни е ъпдейтната. Вече имаме работещ админ панел, в който трябва да се регистрираме.

Проекта трябва да съдържа студенти, преподаватели и дипломни работи. Създаваме всяка таблица(php artisan make: migration create\_students\_table) към базата данни, добавяме колони за име и други данни и правим миграция на таблицата. Php artisan backpack:crud student създава за нас CRUD item - create, read, update and delete и вече имаме контролер, модел, рикуест и ъпдейт на views и routes. Вече можем да добавим студент/преподавател с информация за него. За дипломните работи трябва да създадем таблица articles с title, content и tags. Първо правим миграция за връзка между таблиците. Добавяме в Article модела:

public function tags()

{

return $this->belongsToMany(Teacher::class, 'article\_teachers','article\_id', 'teacher\_id');

}

В Article контролера добавяме:

private function getFieldsData($show = FALSE) {

return [

[

'name'=> 'title',

'label' => 'Title',

'type'=> 'text'

],

[

'name' => 'content',

'label' => 'Content',

'type' => ($show ? "textarea": 'ckeditor'),

],

[ // Select2Multiple = n-n relationship (with pivot table)

'label' => "Teachers",

'type' => ($show ? "select": 'select2\_multiple'),

'name' => 'tags', // the method that defines the relationship in your Model

// optional

'entity' => 'tags', // the method that defines the relationship in your Model

'model' => "App\Models\Teacher", // foreign key model

'attribute' => 'Name', // foreign key attribute that is shown to user

'pivot' => true, // on create&update, do you need to add/delete pivot table entries?

],

[

'label' => "Article Image",

'name' => "image",

'type' => 'image',

'crop' => true, // set to true to allow cropping, false to disable

'aspect\_ratio' => 1, // omit or set to 0 to allow any aspect ratio

]

];

}

Ъпдейтваме сетъп метода и добавяме нов метод в края на контролера:

protected function setupShowOperation()

{

// by default the Show operation will try to show all columns in the db table,

// but we can easily take over, and have full control of what columns are shown,

// by changing this config for the Show operation

$this->crud->set('show.setFromDb', false);

$this->crud->addColumns($this->getFieldsData(TRUE));

}

Имаме осъществена връзка между таблиците.

За качване на изображения към articles задаваме миграция “php artisan make:migration add\_image\_to\_articles\_table --table=articles”:

/\*\*\* Run the migrations.\*\* **@return** void\*/public function up()  
{  
Schema::table('articles', function (Blueprint $table) {  
$table->string('image',255)->nullable();  
});  
}  
  
/\*\*\* Reverse the migrations.\*\* **@return** void\*/public function down()  
{  
Schema::table('articles', function (Blueprint $table) {  
$table->dropColumn('image');  
});  
}

Колоната за изображение е добавена в таблицата в базата данни. Към модела добавяме:

use Illuminate\Support\Str;

use Intervention\Image\ImageManagerStatic as Image;

use Illuminate\Support\Facades\Storage;

…

public static function boot()

{

parent::boot();

static::deleting(function($obj) {

Storage::delete(Str::replaceFirst('storage/','public/', $obj->image));

});

}

public function setImageAttribute($value)

{

$attribute\_name = "image";

// destination path relative to the disk above

$destination\_path = "public/articles";

// if the image was erased

if ($value==null) {

// delete the image from disk

Storage::delete($this->{$attribute\_name});

// set null in the database column

$this->attributes[$attribute\_name] = null;

}

// if a base64 was sent, store it in the db

if (Str::startsWith($value, 'data:image'))

{

// 0. Make the image

$image = Image::make($value)->encode('jpg', 90);

// 1. Generate a filename.

$filename = md5($value.time()).'.jpg';

// 2. Store the image on disk.

Storage::put($destination\_path.'/'.$filename, $image->stream());

// 3. Delete the previous image, if there was one.

Storage::delete(Str::replaceFirst('storage/','public/', $this->{$attribute\_name}));

// 4. Save the public path to the database

// but first, remove "public/" from the path, since we're pointing to it

// from the root folder; that way, what gets saved in the db

// is the public URL (everything that comes after the domain name)

$public\_destination\_path = Str::replaceFirst('public/', 'storage/', $destination\_path);

$this->attributes[$attribute\_name] = $public\_destination\_path.'/'.$filename;

}

}

като първата функция отговаря за изтриване на изображението при изтриване на артикъла, а втората за качване на изображението в правилната директория. Изпълняваме “composer require intervention/image”. Към контролера добавяме:

[

'label' => "Article Image",

'name' => "image",

'type' => 'image',

'crop' => true, // set to true to allow cropping, false to disable

'aspect\_ratio' => 1, // omit or set to 0 to allow any aspect ratio

]

И

protected function setupListOperation()

{

$this->crud->set('show.setFromDb', false);

$this->crud->addColumns($this->getFieldsData(TRUE));

}

Извършваме командата “php artisan storage:link”, за да бъдат файловете видими.

Тъй като полетата за въвеждане на студенти, преподаватели и дипломни работи не могат да останат празни трябва да добавим следното валидиране в request файловете:

public function rules()

{

return [

'title' => 'required|min:5|max:255',

'content' => 'required'

];

}

{

return [

'Name' => 'required|min:5|max:255',

'Major' => 'required|min:5|max:255'

];

}

За потребителската част на система ни трябва да интегрираме темплейт, където ще се преглеждат добавените в админ панела студенти, преподаватели и дипломни работи. Създаваме контролер за основната част на темата, която ще бъде и наша заглавна страница. Добавяме следното съдържание:

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class IndexController extends Controller

{

public function index() {

//Get data from DB in here and pass it to the view

return view('index.index', [

'title' => 'Welcome!',

'text' => '

<hr />

<h3><p>This is a student management system for students and their teachers and articles based on Laravel 8.x. </p></h3>'

]);

}

}

Трябва да добавим Index контролера в routes. Раутингът ни служи да посочим на системата пътя към правилния контролер.

В resources/views/layouts създаваме нов файл, в който копираме съдържанието от основния html файл. Заменяме главното съдържание на файла с @yield(‘content’), за да можем за използваме същия layout(header,footer, etc.) при другите страници.