web разработка

Models

Съдържание

Задача

Създайте два вида потребители - Администратор и Ученик.
Отпечатайте профила на потребителя на страница Профил.
Опечатайте поставените теми за домашно, които Администраторът на сайта е добавил.
Добавете функционалност за регистрация/логин на потребителя.
Всички потребители по подразбиране се добавят като ученици. Има един администратор на сайта, чиято роля е посочена директно в базата данни.

Model
V
C

Model

Моделът отговаря за бизнес логиката на приложението.

Работи с базата данни и информацията в нея.

Методите в моделите обработват потока от информация в приложението - в нужния формат и количество я предоставят на контролера и той от своя страна я прехвърля във видимата част на приложението. Или в обратна посока - поемат постъпващата в приложението потребителска информация и я записват в базата данни. Ако се налага преди това я трансформират.

Връзка с базата данни в Ларавел

Връзка с базата данни

Конфигурационния файл, отговарящ за базата данни е config/database.php

За да свържем приложението с неговата база данни в .env

Попълваме съответните стойности за

DB_DATABASE=laravel_students/името на базата данни/

DB_USERNAME=root/потребител/

DB_PASSWORD=secret/парола, ако има/

Laravel Migration

Laravel Migration

Миграциите са като GitHub за базата данни, позволяващи на екипа от разработчици лесно да променя и споделя структурата на базата данни на приложението.

Миграциите са съпроводени от т. нар. schema builder, който позволява лесно да се построи конструкцията на базата данни.

Ако някога ви се е налагало да добавяте колона към таблица в базата данни , миграциите решават и такива проблеми.

Facade Schema поддържа създаване и промяна на всички видове бази данни, които Ларавел поддържа.

Задача. Направете проект на нужната база данни за приложението, което разработваме

Laravel Migration

За да създадете миграция използвайте командата

php artisan make:migration create_users_table

Тя ще се появи в database/migrations directory.

Всеки файл-миграция съдържа в името си момента, в който е създаден, което позволява на Ларавел да определи реда на миграциите.

Важно! Не изтривайте файловете-миграции, така нарушавате реда, който Ларавел следи!

Laravel Migration

--table и --create опциите, могат да бъдат използване за индикация дали се създава нова таблица или се прави модификация на вече съществуваща таблица.

Тези опции определят и съдържанието на файла-миграция.

Да пробваме -

php artisan make:migration create_users_table --create=users //създаваме таблица

php artisan make:migration add_votes_to_users_table --table=users //добавяме колона към съществуваща таблица

Да разгледаме съдържанието на файла-миграция

Laravel Migration

Миграция за таблица users има по подразбиране в Ларавел.

В нея можем да добавяме колони, ако преценим, че ни трябват.

Нека добавим колона user_role.

Преди да сме мигрирали/ процес при, който миграционния файл създава таблица в базата данни/ можем директно във файла да добавим колоната, която ни трябва.

След миграция на таблицата, за да добавим колона ще трябва да използваме командата

php artisan make:migration add_user_role_to_users_table --table=users

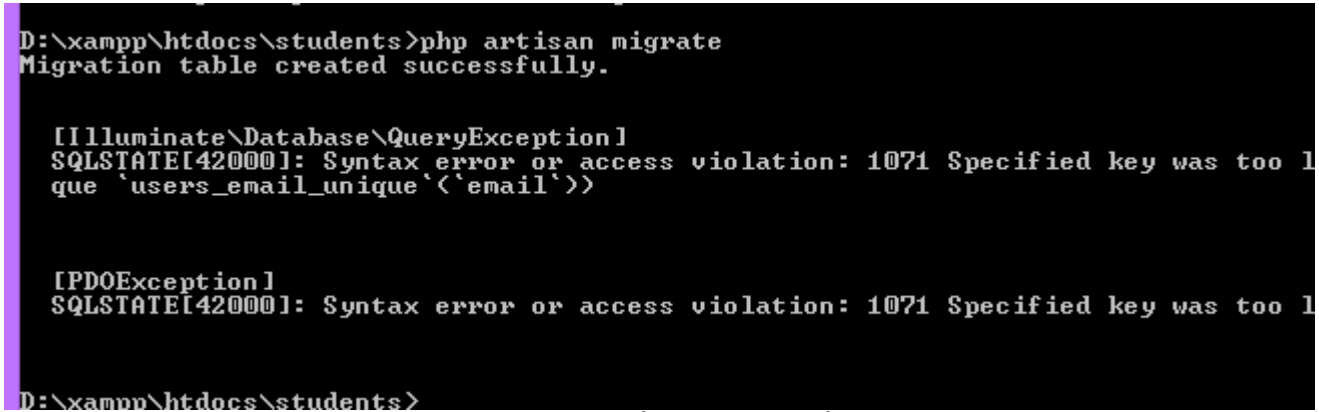
Laravel Migration - да обобщим

Да изпълним следните стъпки за нашето приложение:

1. Създаваме база данни - laravel_students
2. Добавяме в .env нужната информация
3. Добавяме в CreateUsersTable миграционнен файл колона user_role
4. Модел User също е създаден при инсталацията на Ларавел, по късно в ще ,аботим в него.
5. Изпълняваме командата php artisan migrate, която създава таблицата users в базата данни
6. Заедно с нея в базата данни се създават и други таблици - прегледайте ги. Те не бива да се трият.

Laravel Migration - да обобщим

След изпълняването на командата за при Ларавел 5,4 се появява следната грешка



Изпълняваме - migrate:rollback и изтриваме таблиците в базата данни

Laravel Migration

При Ларавел 5,4 трябва да се съобразим със следното

Дължината на индексите и MySQL / MariaDB

По подразбиране Laravel използва utf8mb4, което позволява и съхраняване на "emojis" в базата данни.
Ако използвате версия на MySQL, по-стара от 5.7.7 или MariaDB по-стара от 10.2.2, ще трябва да конфигурирате дължината на стринга по подразбиране, генериран при миграция, с цел MySQL да създаде индекси за тях.

Laravel Migration

Единия от вариантите за това е чрез извикване на метода Schema::defaultStringLength в AppServiceProvider:

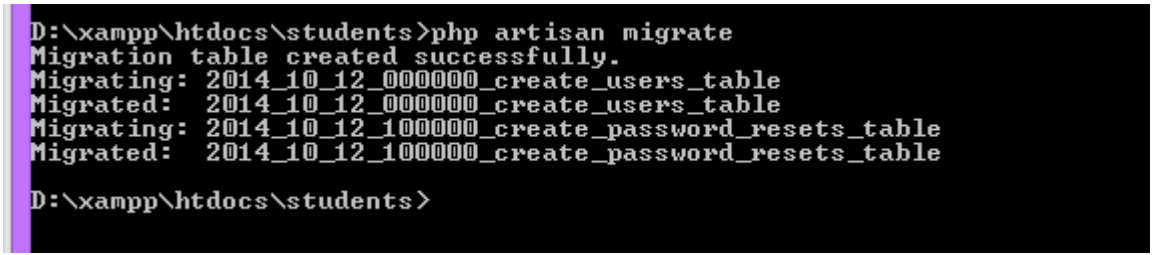
```
use Illuminate\Support\Facades\Schema;

/**
 * Bootstrap any application services.
 *
 * @return void
 */
public function boot()
{
    Schema::defaultStringLength(191);
}
```

Laravel Migration

отново изпълняваме командата

php artisan migrate



Получаваме съобщение за успешна миграция и създаване на таблици.

Laravel Eloquent

Laravel Eloquent

The Eloquent ORM/object relational mapping/, включен в Ларавел, предоставя елегантен и лесен начин за работа с базата данни.

Всяка таблица от базата данни притежава свой Модел, който се използва за взаимодействие с нея.

Моделите ви позволват да четете, променяте и добавяте нова информация в таблиците в БД.

Преди всичко се уверете, че сте конфигурирали връзката с базата данни в config/database.php.

За повече информация относно конфигурирането на вашата база данни направете справка с документацията на Ларавел.

Laravel Eloquent

Като начало да създадем Eloquent модел.

Моделите обикновено се разполагат в директорията app.

Всички Eloquent модели наследяват Iluminate\Database\Eloquent\Model class.

Най-лесния начин да създадете модел е чрез командата

php artisan make:model ИметоНаМодела

Laravel Eloquent

Ако искате да генерирате и файл миграция за базата данни, може да добавите опцията --migration или -m option:

php artisan make:model ИметоНаМодела --migration

php artisan make:model ИметоНаМодела -m

За допълнителна информация - https://laravel.com/docs/5.4/eloquent

Laravel Eloquent

За допълнителна информация - https://laravel.com/docs/5.4/eloquent

Прочетете я, тъй като Eloquent ORM приема по подразбиране, неща с които трябва да се съобразяваме при разработка.

Ако трябва да променяте нещо - в имена на таблици, специфични колони, друго, трябва да го направите по начин, който Eloquent ORM ще разбере правилно.

Задача

Като използвате вашия проект за база данни на приложението, създайте

1. Модел и миграция Lecture, Homework и Profile
2. Добавете в миграцията нужните колони
3. Мигрирайте таблицата в базата данни

За описание на колоните в миграцията следвайте написаното в CreateUsersTable миграця-файл и правете справка с

https://laravel.com/docs/5.4/migrations

a. Available Column Types
b. Column Modifiers
c. Creating Indexes
d. Foreign Key Constraints

protected $fillable

Protected $fillable

Всеки модел има свойство $fillable.

Ако искаме да разрешим на потребителя да работи със записите в определени колони въвеждаме имената на колоните като масив.

Например в модела Users

```
protected $fillable = [

        'name', 'email', 'password',

        ];
```

Eloquent Relationships

Eloquent Relationships

Таблиците в базите данни много често са свързани по между си.
Например един пост може да има много коментарии, или една поръчка е свързана с потребителя, който я е направил.

Eloquent прави работата със свързаните таблици много лесна и поддържа различните видове връзки -

- One To One
- One To Many
- Many To Many
- Has Many Through
- Polymorphic Relations
- Many To Many Polymorphic Relations

Eloquent Relationships

Връзките се декларират като методи в класовете на Eloquent model.

Също както и моделите, тези връзки са мощно средство за генериране на заявки към базата данни. Дефинирането на връзките като методи ни позволява и да навързваме методи - да прилагаме метод върху резултата от предишния метод във веригата.

```
$user->posts()->where('active', 1)->get();
```

//кодът ще върне постовете на конкретния потребител, за които стойността в колоната active е 1

Eloquent Relationships

One To One

Пример за такава връзка в нашето приложение е - всеки потребител има по един профил.
Информацията е разпределена в две таблици - Users и Profile.
users_id в Profile отговаря на id в Users.

За да дефинираме тази връзка разполагаме profile() метод в модела User
Методът profile() тряба да извика hasOne функцията и да върне резултатът от нея.

Eloquent Relationships

```
namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Get the profile record associated with the user.
     */
    public function profile()
    {
        return $this->hasOne('App\Profile');
    }
}
```

Eloquent Relationships

Аргументът подаден на hasOne е името на свързания модел.

След като връзката между моделите е дефинирана извикваме съответния запиі от базата данни, използвайки динамичните свойства на Eloquent. Тези динамични свойства позволяват да извикваме методите като свойства декарирани в съответния модел.

```
$profile = User::find(1)->profile;
```

Eloquent Relationships

Дефиниране на обратната връзка между моделите

Сега може да достъпим профила на конкретния потребител /например, който е логнат в приложението/.
Да дефинираме и връзка в Profile модела, която да сочи към потребителя, който притежава този профил.

Eloquent Relationships

Дефинираме обратната връзка чрез методът belongsTo:

```
namespace App;

use Illuminate\Database\Eloquent\Model;

class Profile extends Model
{
    /**
     * Get the user that owns the profile.
     */
    public function user()
    {
        return $this->belongsTo('App\User');
    }
}
```

Eloquent Relationships

One To Many

Един модел е свързан/притежава/ неограничен брой други модели.

Например - един пост има неограничен брой коментари.

Една лекция ще има много домашни /като тема, описание, краен срок за качване на домашни и т.н./.

Eloquent Relationships

Дефинираме връзката чрез съответната функция в модела -

```
class Lecture extends Model
{
    /**
     * Get the homeworks for the lecture.
     */
    public function homeworks()
    {
        return $this->hasMany('App\Homework');
    }
}
```

Запомнете - Eloquent автоматично ще определи колоната с външен ключ в модела Homework. По конвенция, Eloquent ще превърне вземе името на съотвения модели ще добави _id и ще търси тази колона за външен ключ.
В нашия случай ще търси колоната **homework_id**.

Ако се налага да използвате имена на колони и таблици, различни от тези по подразбиране, направете справка с документацията как да направите това, така че Eloquent да разбере правилно различията!

Eloquent Relationships

След като сме дефинирали връзката, можем да достъпим колекцията от домашни чрез **homeworks** свойството на модела.
Благодарение на динамичните свойства на Eloquent - достъпваме методите, дефиниращи връзката като свойства на съответните модели.

```
$homeworks = App\Lecture::find(1)->homeworks;

foreach ($homeworks as $homework) {
   //
}
```

Eloquent Relationships

И тъй като връзките между моделите служат за формиране на заявки, може да продължите веригата от функции като добавите ограничения към колекцията домашни, които искате да получите -

```
$homeworks = App\Lecture::find(1)->homeworks()->where('column_name', 'foo')->first();
```

//where('column_name', 'foo') съответства на where column_name=foo

Eloquent Relationships

One To Many (обратна връзка)

Вече имаме достъп до домашните на всяка лекция. Следва да дефинираме обратната връзка - да позволим достъпа до лекцията, към която всяко домашно е свързано.

За да дефинираме обратната на hasMany връзка, извикваме метода belongsTo в модела, който искаме да свържем /Homework/:

Eloquent Relationships

```
namespace App;

use Illuminate\Database\Eloquent\Model;

class Homework extends Model
{
    /**
     * Get the lecture that owns the homework.
     */
    public function lecture()//в единствено число!!
    {
        return $this->belongsTo('App\Lecturet');
    }
}
```

Eloquent Relationships

Сега имаме вече достъп до лекцията, чрез lecture динамичното свойство на homework.

```
$homework = App\Homework::find(1);

echo $homework->lecture->name;
```

Eloquent ще свърже lecture_id от Homework модела с id от Lecture модела.
Eloquent определя външния ключ като добавя към името на модела посочен във връзката-метод _id -

Lecture -> lecture + _id = lecture_id

Задача

1. Допълнете липсващите модели и методи, дефиниращи всички връзки между моделите
    a. User
    b. Profile
    c. Homework
    d. Lecture
    e. UserLectureHomework
2. Разрешете колоните, в които потребителите ще работят със записите.