web разработка

Models
Eloquent

Съдържание

Задача

- Направете проект на база данни за проекта
- Добавете базата данни в phpMyAdmin
- Създайте два вида потребители - Администратор и Ученик.
- Отпечатайте профила на потребителя на страница Профил.
- Отпечатайте 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.

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/5.5/eloquent

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 миграция-файл и правете справка с

https://laravel.com/docs/5.5/migrations

Available Column Types

a. Column Modifiers
b. Creating Indexes
c. 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',
        ];
```

Задача

Добавете protected $fillable свойство в създадените от вас модели.

Задайте стойности според това, в кои от колоните на съответните таблици ще записвате или променяте данни.

Eloquent Relationships

Eloquent Relationships

Таблиците в базите данни много често са свързани по между си.
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.
user_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 запис от базата данни, използвайки динамичните свойства на Eloquent. Тези динамични свойства позволяват да извикваме методите като *свойства декларирани в съответния модел.*

$profile = User::find(1)->profile; //обект от клас … да проверим

Eloquent Relationships

Или извикан като метод profile() – да достъпим свойствата на свързания клас Profile.

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

Eloquent Relationships

Дефиниране на обратната връзка между моделите

Съществува възможност да достъпим профила на конкретния потребител /например, който е логнат в приложението/.
За целта нека 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

Получаваме възможност да

```
$profile_user = Profile::find($id)->user()->username;
```

достъпим всяко от свойствата на класа обекта от клас User през свързания обект от клас Profile.

```
$user_first_name = User::find($id)->profile()->first_name
```

Достъпим всяко от свойствата на обекта от клас Profile през свързания обект от клас User

Eloquent Relationships

One To Many

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

Например –

Един пост има неограничен брой коментари.

Една лекция има много домашни.

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 вземе името на съотвения модели и ще добави _id - ще търси колона с такова име 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

//first() съответства на LIMIT 1
```

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\Lecture');
    }
}
```

Eloquent Relationships

Сега имаме достъп и 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 методи дефиниращи всички връзки между моделите
    a. User
    b. Profile
    c. Module
    d. Lecture
2. Разрешете колоните, в които потребителите ще работят със записите.
3. Приложението ви трябва да притежава страници
    - Профил на потребителя
    - Списък с модули
    - Списък с лекции към всеки модул

    Допълнете липсващите вюта и методи в контролерите /ако има такива/
4. Отпечатайте информация на съответните страници
    - Профил на потребителя
    - Списък с модулите
    - Списък с лекции към модул с ид 1

Задача

6. Добавете модел и миграция за Курс. Приемаме, че в нашите курсове няма модули, принадлежащи на няколко курса.
7. Дефинирайте връзките между Курс и Модул.
8. Създайте контролер за Курс и отпечатайте във вю списък с наличните курсове.
9. Отпечатайте модулите, които принадлежат на курс с ид=1
10. Списъците – курсове, модули, лекции да се достъпват с бутони.
11. Каква е връзката ниво-курсист. Дефинирайте в съответните модели с помощта на Laravel Eloquent. Отпечатайте курсовете, в които участва курсист с ид=1. /Един курсист може да участва в няколко курса/.

**Забележка* Данните въвеждайте ръчно в БД.