Lista https://www.youtube.com/playlist?list=PLhCiuvlix-rT96yLQTTr3-eTZl6sbhVte

Añadir la debug bar

<https://packagist.org/packages/barryvdh/laravel-debugbar>

Relación 1 a 1

User.php

 public *function* profile()

    {

        return $this->hasOne(*Profile*::class);

    }

Un usuario tiene un perfil. No hago la inversa porque no necesito acceder al usuario a través del perfil

*Schema*::create('profiles', *function* (*Blueprint* $table) {

            $table->id();

            $table->bigInteger('user\_id')->unsigned();

            $table->string('instagram')->nullable();

            $table->string('github')->nullable();

            $table->string('web')->nullable();

            $table->timestamps();

            $table->foreign('user\_id')->references('id')->on('users')

                ->onDelete('cascade')

                ->onUpdate('cascade');

        });

Cuando borre el usuario me borras en cascada el perfil y si lo actualizo también actualizo el perfil

Relación Uno a Muchos

Un nivel tiene muchos usuarios Level.php Ej Nivel Oro hay 100 usuarios

public *function* users()

    {

        return $this->hasMany(*User*::class);

    }

Me interesa la inversa poruq quiero saber cual es el nivel individual de cada usuario User.php

public *function* level()

    {

        return $this->belongsTo(*Level*::class);

    }

Para incluir un campo o una foreign key sin borrar toda la base de datos

php artisan make:migration add\_level\_id\_at\_users

  \*/

    public *function* up()

    {

*Schema*::table('users', *function*(*Blueprint* $table){

            $table->bigInteger('level\_id')

            ->unsigned()

            ->nullable()

            ->after('id');

            $table->foreign('level\_id')->references('id')->on('levels')

                ->onDelete('set null')

                ->onUpdate('cascade');

        });

    }

Imagina que ya es una bd en producción y creas la tabla levels. Los usuarios que ya están no tienen campo level\_id entonces para que no pete tienes que ponerlo nulo.

Además por el mismo principio cuando borres un nivel no borres los usuarios sino pon el null el campo con el set null

**Relación Muchos a Muchos**

Creo las tablas normales en este caso relaciono Users y Groups Los usuarios pueden tener muchos grupos y pertenecer a muchos grupos. Para esto se necesita una tabla pivot

php artisan make:migration create\_group\_user\_table

Creo solo la tabla y siempre en singular por los modelos, minúsculas y ordenadas alfabéticamente

Tabla Pivot group\_user

*Schema*::create('group\_user', *function* (*Blueprint* $table) {

            $table->id();

            $table->bigInteger('group\_id')->unsigned();

            $table->bigInteger('user\_id')->unsigned();

            $table->timestamps();

            $table->foreign('group\_id')->references('id')->on('groups')

                ->onDelete('cascade')

                ->onUpdate('cascade');

                $table->foreign('user\_id')->references('id')->on('users')

                ->onDelete('cascade')

                ->onUpdate('cascade');

        });

Luego en los modelos relacionados

Group

public *function* users()

    {

        return $this->belongsToMany(*User*::class)->withTimestamps();

    }

User

public *function* groups()

    {

        return $this->belongsToMany(*Group*::class)->withTimestamps();

    }

Ojo las tablas por defecto incluyen el created\_at y updated\_at porque en las migraciones por defecto se incluye los timestamps.

*Schema*::create('groups', *function* (*Blueprint* $table) {

            $table->id();

            $table->string('name');

            $table->timestamps();

        });

Así que si no quieres que cada vez que se cree un registro aparezca estos campos en blanco le pones el ->withTimestamps()

**RELACION UNO A UNO A TRAVES**

Quiero acceder a una tabla Localizaciones que almacena la ciudad desde usuarios directamente. Poder consultar la ciudad del usario

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Creo la tabla locations y la relaciono con profiles porque las localizaciones dependen del perfil y el perfil del usuario

Profile migration

*Schema*::create('locations', *function* (*Blueprint* $table) {

            $table->id();

            $table->bigInteger('profile\_id')->unsigned();

            $table->string('country');

            $table->timestamps();

            $table->foreign('profile\_id')->references('id')->on('profiles')

            ->onDelete('cascade')

            ->onUpdate('cascade');

        });

Y ahora quiero relacionar Usuarios con Locations

User

 public *function* location()

    {

        return $this->hasOneThrough(*Location*::class, *Profile*::class);

    }

El usuario tiene una location a través del perfil. Así se haría y luego profile que es quien esta relacionada 1-1 pues normal

Profile

    public *function* location()

    {

        return $this->hasOne(*Location*::class);

    }

Creo las tablas Categories, posts, videos, comments, images, tags y taggables solo como migración

**Tanto Images como Comments como Taggables son POLIMORFICAS por ejemplo Images**

public *function* up()

    {

*Schema*::create('images', *function* (*Blueprint* $table) {

            $table->id();

            $table->string('url');

            $table->morphs('imageable');

            $table->timestamps();

        });

    }

El morphs crea dos campos:

Imageable\_id e imageable\_type

En uno guarda el id y en otro la entidad o modelo

Con lo cual images puede ser usada en perfil, usuario, video etc etc al igual que los comentarios: puede haber comentarios en videos, en posts etc

Y en los modelos se pondría de esta forma

    public *function* comments()

    {

        return $this->morphMany(*Comment*::class, 'commentable');

    }

    public *function* image()

    {

        return $this->morphOne(*Image*::class, 'imageable');

    }

    public *function* tags()

    {

        return $this->morphToMany(*Tag*::class, 'taggable');

    }

Uno a uno morphOne y uno a muchos morphMany Polimorfica y muchos a muchos morphToMany

Siempre tiene que nombrarse el morph en singular y con la terminación able

El modelo EER quedaría así

Diagrama

Descripción generada automáticamente

Relaciones resumen

Category

Una Categoria tiene muchos posts y muchos videos

*class* Category extends *Model*

{

    use *HasFactory*;

    public *function* posts()

    {

        return $this->hasMany(*Post*::class);

    }

    public *function* videos()

    {

        return $this->hasMany(*Video*::class);

    }

}

Diagrama

Descripción generada automáticamente

Comments

Comentarios es una tabla polimórfica y lo se por el nombre del método y por el morphTo y también un comentario pertenece a un usuario

Diagrama

Descripción generada automáticamente

*class* Comment extends *Model*

{

    use *HasFactory*;

    public *function* commentable()

    {

        return $this->morphTo();

    }

    public *function* user()

    {

        return $this->belongsTo(*User*::class);

    }

}

Grupos

Un grupo puede tener muchos usuarios

Diagrama

Descripción generada automáticamente

*class* Group extends *Model*

{

    use *HasFactory*;

    public *function* users()

    {

        return $this->belongsToMany(*User*::class)->withTimestamps();

    }

}

Diagrama

Descripción generada automáticamente

Imagen

Es una tabla polimórfica. En este caso es uno a uno, y como definimos si es uno a uno o uno a muchos? En la entidad que lo una por ejemplo

Post

public *function* image()

    {

        return $this->morphOne(*Image*::class, 'imageable');

    }

Interfaz de usuario gráfica, Aplicación, Excel

Descripción generada automáticamente

*class* Image extends *Model*

{

    use *HasFactory*;

    public *function* imageable()

    {

        return $this->morphTo();

    }

}

Level

Un nivel tiene muchos usuarios y muchos posts y videos a través de Usuarios

Diagrama

Descripción generada automáticamente

*class* Level extends *Model*

{

    use *HasFactory*;

    public *function* users()

    {

        return $this->hasMany(*User*::class);

    }

    public *function* posts()

    {

        return $this->hasManyThrough(*Post*::class, *User*::class);

    }

    public *function* videos()

    {

        return $this->hasManyThrough(*Video*::class, *User*::class);

    }

}

Location

No es necesario crear la relación porque nunca voy a necesitar saber un perfil a través de la localización. Si lo necesitara lo haría

*class* Location extends *Model*

{

    use *HasFactory*;

}

Post

Un post pertenece a un usuario

Un post pertenece a una categoría

Tiene muchos comentarios

Tiene UNA imagen

Tiene muchas etiquetas

Diagrama

Descripción generada automáticamente

*class* Post extends *Model*

{

    use *HasFactory*;

    public *function* user()

    {

        return $this->belongsTo(*User*::class);

    }

    public *function* category()

    {

        return $this->belongsTo(*Category*::class);

    }

    public *function* comments()

    {

        return $this->morphMany(*Comment*::class, 'commentable');

    }

    public *function* image()

    {

        return $this->morphOne(*Image*::class, 'imageable');

    }

    public *function* tags()

    {

        return $this->morphToMany(*Tag*::class, 'taggable');

    }

}

Profile

Tiene una localización y como no entramos desde el perfil al usuario por eso no hace falta la inversa con user

Diagrama

Descripción generada automáticamente

*class* Profile extends *Model*

{

    use *HasFactory*;

    public *function* location()

    {

        return $this->hasOne(*Location*::class);

    }

}

Tag

Tiene muchos posts y videos

*class* Tag extends *Model*

{

    use *HasFactory*;

    public *function* posts()

    {

        return $this->morphedByMany(*Post*::class, 'taggable');

    }

    public *function* videos()

    {

        return $this->morphedByMany(*Video*::class, 'taggable');

    }

}

Diagrama

Descripción generada automáticamente

User

Tiene un perfil Tiene una localización a través de perfil Tiene muchos comentarios

Pertenece a un nivel Tiene muchos posts Tiene una imagen

Puede pertenecer a varios grupos Tiene muchos videos

public *function* profile()

    {

        return $this->hasOne(*Profile*::class);

    }

    public *function* level()

    {

        return $this->belongsTo(*Level*::class);

    }

    public *function* groups()

    {

        return $this->belongsToMany(*Group*::class)->withTimestamps();

    }

    public *function* location()

    {

        return $this->hasOneThrough(*Location*::class, *Profile*::class);

    }

    public *function* posts()

    {

        return $this->hasMany(*Post*::class);

    }

    public *function* videos()

    {

        return $this->hasMany(*Video*::class);

    }

    public *function* comments()

    {

        return $this->hasMany(*Comment*::class);

    }

    public *function* image()

    {

        return $this->morphOne(*Image*::class, 'imageable');

    }

Video

Prácticamente igual que user

Tiene un usuario Tiene una imagen

Pertenece a una categoria Tiene muchas etiquetas

Tiene muchos comentarios

*class* Video extends *Model*

{

    use *HasFactory*;

    public *function* user()

    {

        return $this->belongsTo(*User*::class);

    }

    public *function* category()

    {

        return $this->belongsTo(*Category*::class);

    }

    public *function* comments()

    {

        return $this->morphMany(*Comment*::class, 'commentable');

    }

    public *function* image()

    {

        return $this->morphOne(*Image*::class, 'imageable');

    }

    public *function* tags()

    {

        return $this->morphToMany(*Tag*::class, 'taggable');

    }

}