Inés Larrañaga Fernández de Pinedo	CSC Jesuitas - Logroño
	DWES

Table of Contents

Project structure and sail

We are going to create a new laravel project using artisan:

```
composer create-project --prefer-dist laravel/laravel laravel_changeorg
```

After, go to the root of your application using the command line and launch a new container:

```
./vendor/bin/sail up -d
```

Now, if you open your `Docker desktop` application, you should see the container running:

Database model

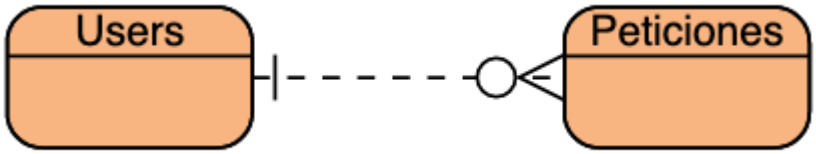
These are going to be our main entities for our Changeorg application:

- Users
- Peticiones
- Categorias

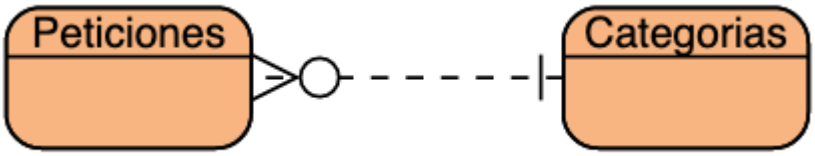
ManyToOne relationships

There are two relationships of this type:

1. Peticiones-Users: A single Peticione belongs to a single User. But a User can have many Peticiones created by himself. That means in a relational DB that inside of `peticiones` table there will be a `user_id` field that will be a FK to the `users` table PK `id`.



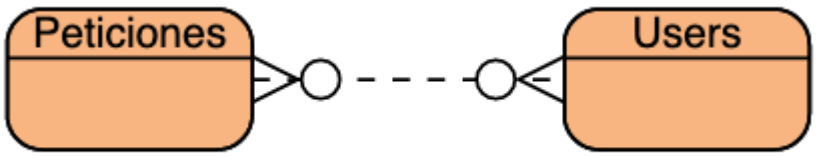
2. Categorias-Peticiones: A single Peticione belongs to a single Category. But a Category can have many Peticiones. That means in a relational DB that inside of `peticiones` there will be a field called `categoria_id`, that will be a FK to the `categorias` table PK `id`.



ManyToMany relationship

The relation between Peticiones and users is a N-M relationship. That means, that we should create and intermediate table in order to hold which user signs which "Peticione":

- A User can sign many Peticiones
- A Peticione can be signed by many Users



DB Migration

We are going to create some migration files in order to reproduce the mentioned entities and relationships. Using the migration capability will allow having the DB schema history updated, and you can recreate it at any time or jump to any point from its history.

If you go into database/migrations folder theare are already some migrations comming by default.

Now we are going to create a new migration file in order to include the creation of our `peticiones` table. In order to create the migration file:

```
php artisan make:migration create_peticiones_table
```

After modify it according to our DB design:

```
public function up()
{
        Schema::create('peticiones', function (Blueprint $table) {
            $table->id();
            $table->string('titulo', 255);
            $table->text('descripcion');
            $table->text('destinatario');
            $table->integer('firmantes');
            $table->enum('estado', ['aceptada', 'pendiente']);
            $table->foreignId('user_id');
            $table->foreignId('categoria_id');
            $table->string('image', 255, );
            $table->timestamps();
        });
}
```

In order the migration command to success you need to setup you `DB_HOST` variable like this:

```
DB_HOST=127.0.0.1
```

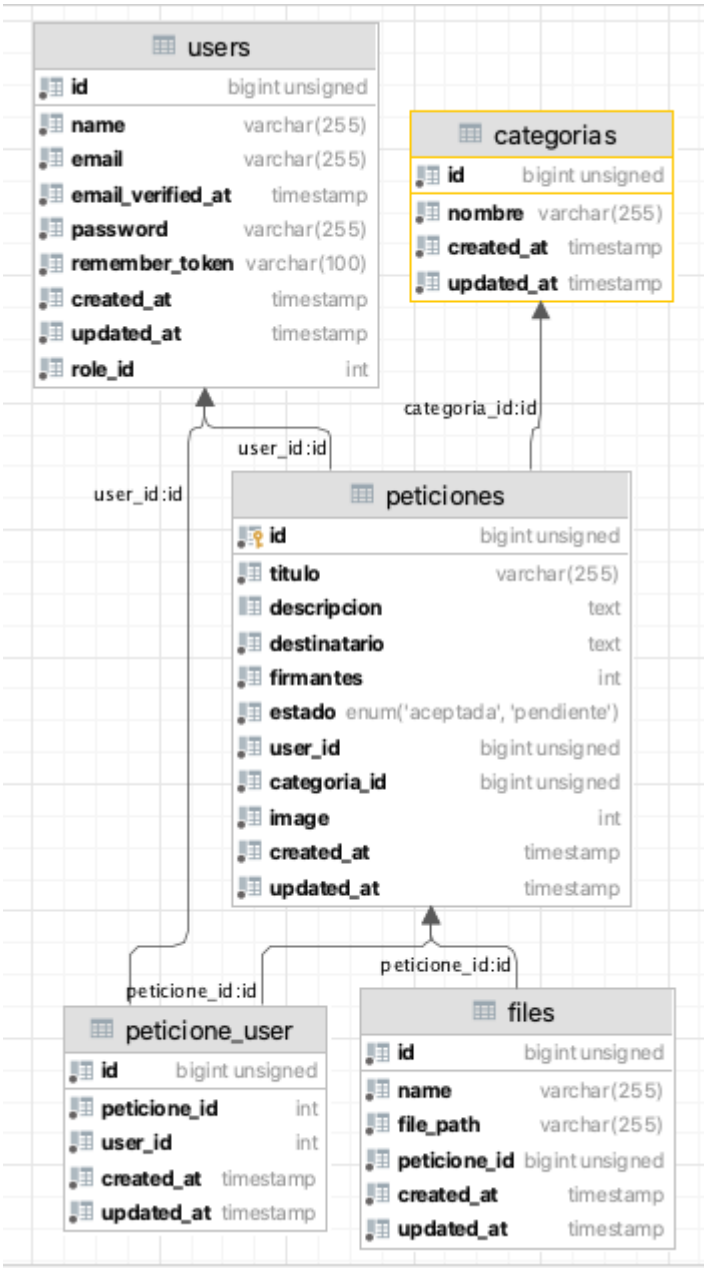
Now lauch the migration tool:

```
php artisan migrate
```

> Exercise: Create the corresponding migration file and launch it in order to create next `categorias` table.
>
> Again, follow the same steps for the pivot table `peticione_user` where `peticione_id` and `user_id` should be FK.

Revert the `DB_HOST` config value to the previous value.

Now, if you see the created DB schema you should obtain something similar to this:



Application environment variables setup

Check that you have properly defined the variables defined inside `.env` file:

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:0hysUoGl4s89D4p+MVkfYTfXCqnSg2mWNSWq9Lu/BY4=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug

DB_CONNECTION=mysql
DB_HOST=mysql
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=sail
DB_PASSWORD=password
```

Layout and home

If you open the application through the explorer you will find a default 'Laravel' page. We are going to enter a new layout into our application.

In order to do so, you should have, for instance, in VSC, the change.org web page you designed using bootstrap framework, and our laravel project opened in `Phpstorm`.

In order to do so, the first thing we should add into our routing file (`routes/web.php`) is what should be loaded for the `home` path:

```
Route::get('/', [\App\Http\Controllers\PagesController::class, 'home']);
```

We are going to create a `PagesController` class using artisan, in order to manage the load of static pages:

```
php artisan make:controller PagesController
```

And inside of it:

```
    public function home()
    {
        return view('pages.home');
    }
```

This means that inside of our templates folder `resources/views/pages` there should be a `home.blade.php` with the content of our home page. Create the file and test it in order to obtain whatever you write on your view.

In any web page, we can have different layout depending, for instance, if you are on the public side or the admin side.

We are going to create an specific layout for the public section and reuse it in every page a normal user can have. So, create a `resources/views/layouts` folder, and inside the corresponding file `public.blade.php`:

```
...
    <script async src="{{asset('js/11391265293.js')}}"></script>
...
<div id="content">
        @yield('content')
</div>
...
```

Do not forget to add the corresponding assets (js,css...) into your `public/` folder.

Now, let,s create the `pages/home.blade.php` with the static content of the page:

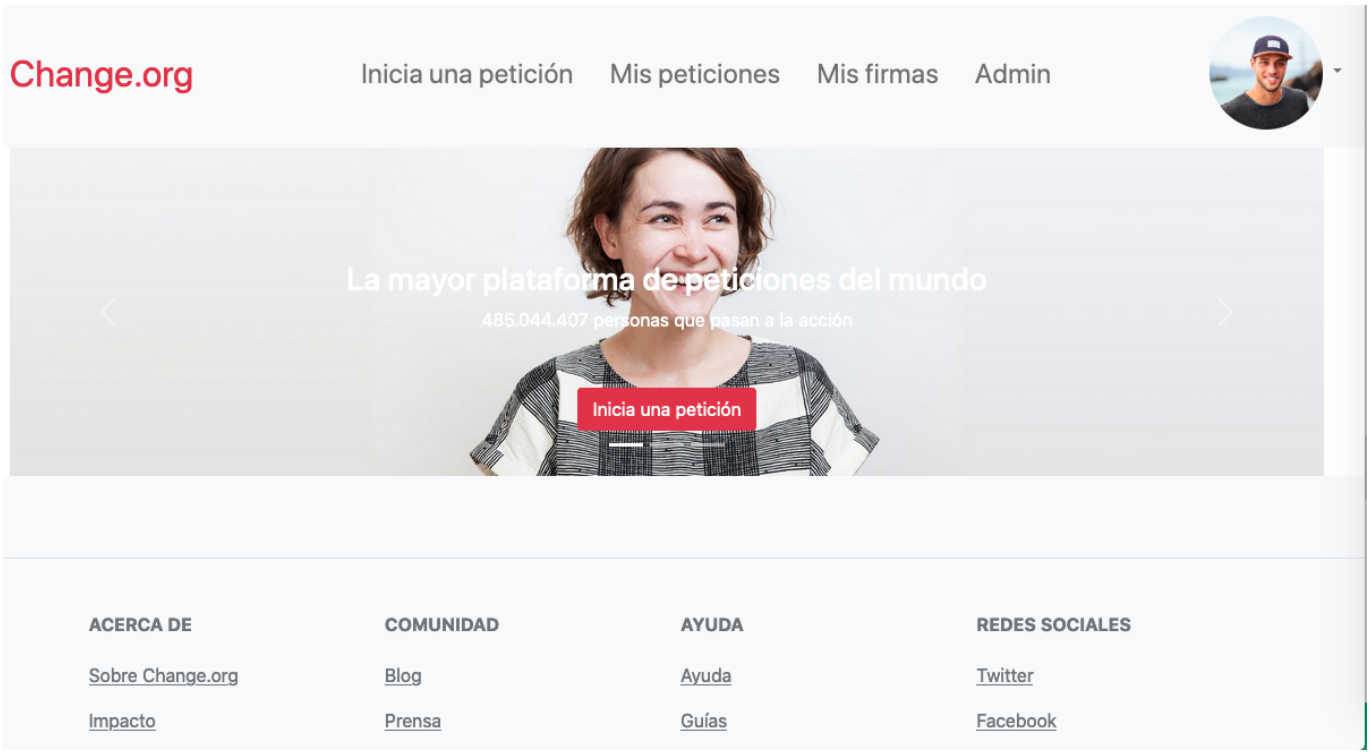
```
@extends('layouts.public')

@section('content')

<div class="home-index" data-view="home/index" data-
userHasSignedBannerPetition=""
    data-fetch_summary="{"featuredTopics":
{"collection":"FeaturedTopicCollection","params":{"locale":"es-ES"}},"quotes":
{"collection":"ManagedListQuoteCollection","params":{"locale":"es-
ES"}},"featuredVictories":{"collection":"FeaturedVictoriesCollection","params":
{"locale":"es-ES","country_code":"ES"}}}">
    <div class="emergency_banner" data-view="components/emergency_banner" data-
user_has_signed_banner_petition=""
  ...
  ...
    </div>
</div>


@endsection
```

The result could be:



Model and controllers creation

If you open de `app/Models` folder you will see that the User model is given to you. Through the command line create the 3 missing models for `Categoria`, `Peticione` and `File`.

```
php artisan make:model Peticione -c
```

If you include the `-c` parameter, apart from the model file, it will include the Controller file on the `app/Http/Controllers` folder.

> On your own, knowing the kind of relationships that we have between entities, include those relationships (bidirectionals) in the models. For instance, the relationship 1-N between `Peticione` and `Categoria`

```
class Peticione extends Model
{
...
    public function categoria(){
        return $this->belongsTo('App\Models\User');
    }
...
}

class Categoria extends Model
{
    use HasFactory;
    public function peticiones(){
        return $this->hasMany('App\Models\Peticione');
    }
}
```

Also define the $fillable attributes of the models. For instance, for `Peticione` model:

```
    protected $fillable = ['titulo', 'descripcion', 'destinatario',
'firmantes', 'estado'];
```

> Complete the same for the rest of the models.