* Clonar Repositorio Laravel
  + No se suben todas las carpetas por seguridad y provoca que el proyecto no arranque bien. Se deben hacer configuraciones adicionales.
  + No instala la carpeta vendor
    - Se debe instalar composer: lee composer.Json para hacer la instalación
    - Instalar npm: Instala node modules
    - Crear el archivo .env: Copiar el de Ejemplo y registrar la información correspondiente.
    - Se debe generar la Key del proyecto. Se genera a través de comando artisan: php artisan key:generate.
    - Realizar las migraciones
  + <https://www.youtube.com/watch?v=EdZ0hQtrfEU>
* Comandos Artisan
  + /\*\*
  + \* Run the migrations.
  + \* php artisan make:migration create\_cursos\_table => Crea una nueva tabla.
  + \* php artisan migrate => Ejecuta las migraciones pendientes.
  + \* php artisan migrate:fresh => Elimina todas las tablas (SIN EJECUTAR método DOWN) y las vuelve a crear (Ejecuta Metodo UP). (Perdida de registros).
  + \* php artisan migrate:refresh =>  Elimina todas las tablas (EJECUTA método DOWN) y las vuelve a crear (Ejecuta Metodo UP).
  + \* php artisan make:migration add\_tipodoc\_to\_users\_table => Convención para agregar una columna sin borrar datos
  + \* php artisan make:migration cambiar\_propiedades\_to\_users\_table  => Convención para modificar propiedades de una columna.
  + \* php artisan migrate:reset: Elimina todas las tablas que se han creado actualmente en la base de datos, sin correr nuevamente las migraciones (Es decir sin ejecutar el método UP) y borra la tabla de migraciones.
  + php artisan db:seed => Ejecuta lo definido en el archivo /Database/seeders/DatabaseSeeder.php
  + php artisan migrate:fresh –seed => Combina Fresh con db:Seed
  + php artisan make:seeder CursoSeeder => Crea un archivo de Seeder independiente en el cual crear el código personalizado de Seeder para cada modelo
  + php artisan make:factory –model=Curso CursoFactory => Crea un archivo para crear Datos tipo Seeder a través de Factory. El achivo se crea en Database/Factories
  + php artisan make:request StoreCurso => Crea un FormRequest
  + php artisan make:middleware CheckAge
* Base de Datos (Migraciones)
  + Schema::Create
    - * UP:

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

$table->id();

$table->string('name');

$table->text('descripcion');

$table->timestamps();

});

* + - * Down
        + Schema::dropIfExists('cursos');
  + Schema::Table
    - Add
      * UP: $table->unsignedTinyInteger('tipodoc')->nullable()->after('avatar');
      * Down: $table->dropColumn('tipodoc');
    - Modify
      * UP: $table->string('name', 150)->nullable()->change();
      * Down: $table->string('name', 255)->nullable(false)->change();
  + Guardar registros (desde tinker)
    - Relaciones de Muchos a Muchos
      * Attach, Detach y Sync son exclusivos para las relaciones de muchos a muchos
        + Agregar

$user->roles()->attach(1)

* + - * + Agregar varios a la vez

$user->roles()->attach([1, 2, 3])

* + - * + Quitar

$user->roles()->detach(1)

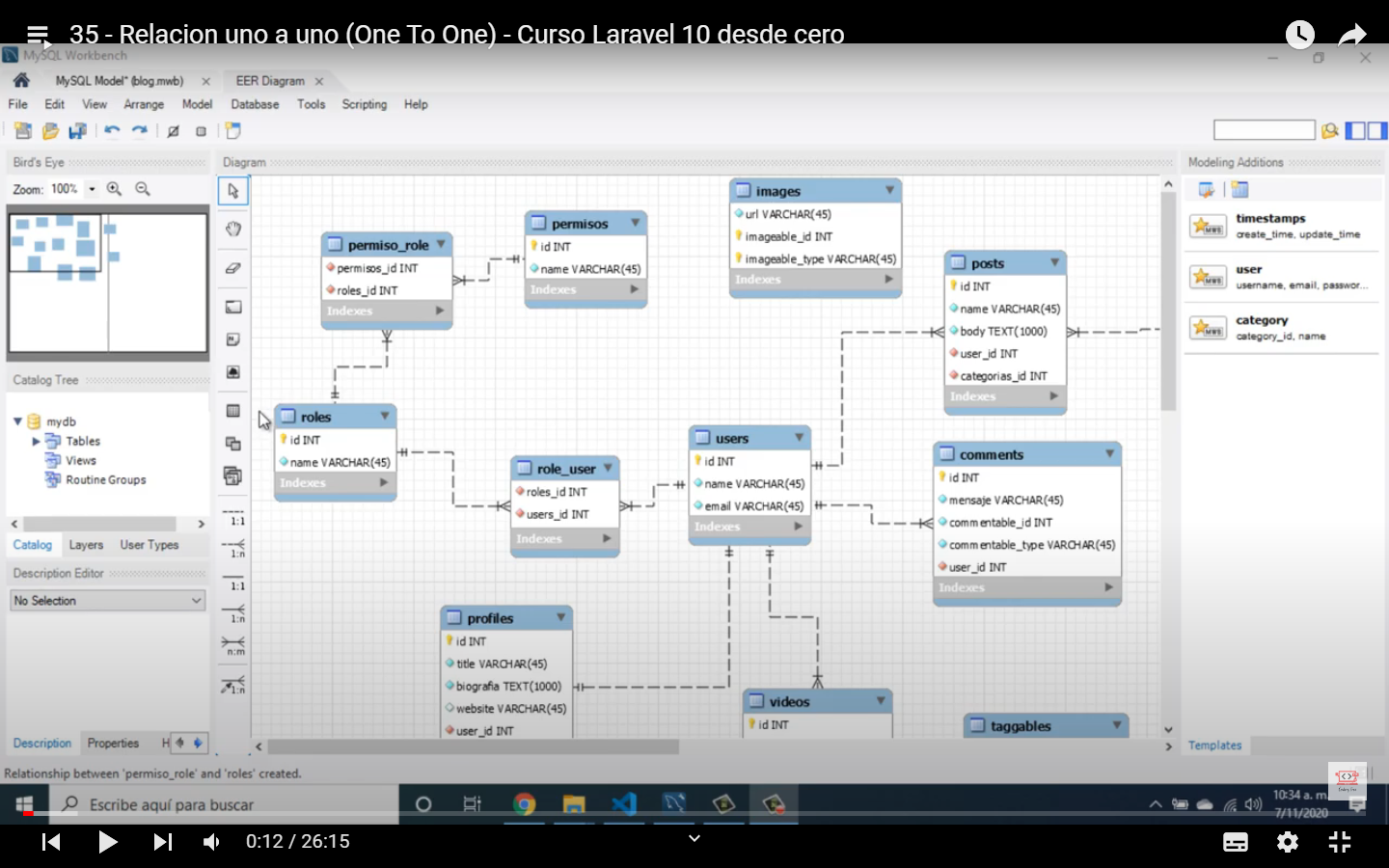
* + - * + Quitar varios a la vez

$user->roles()->detach([1,2,3])

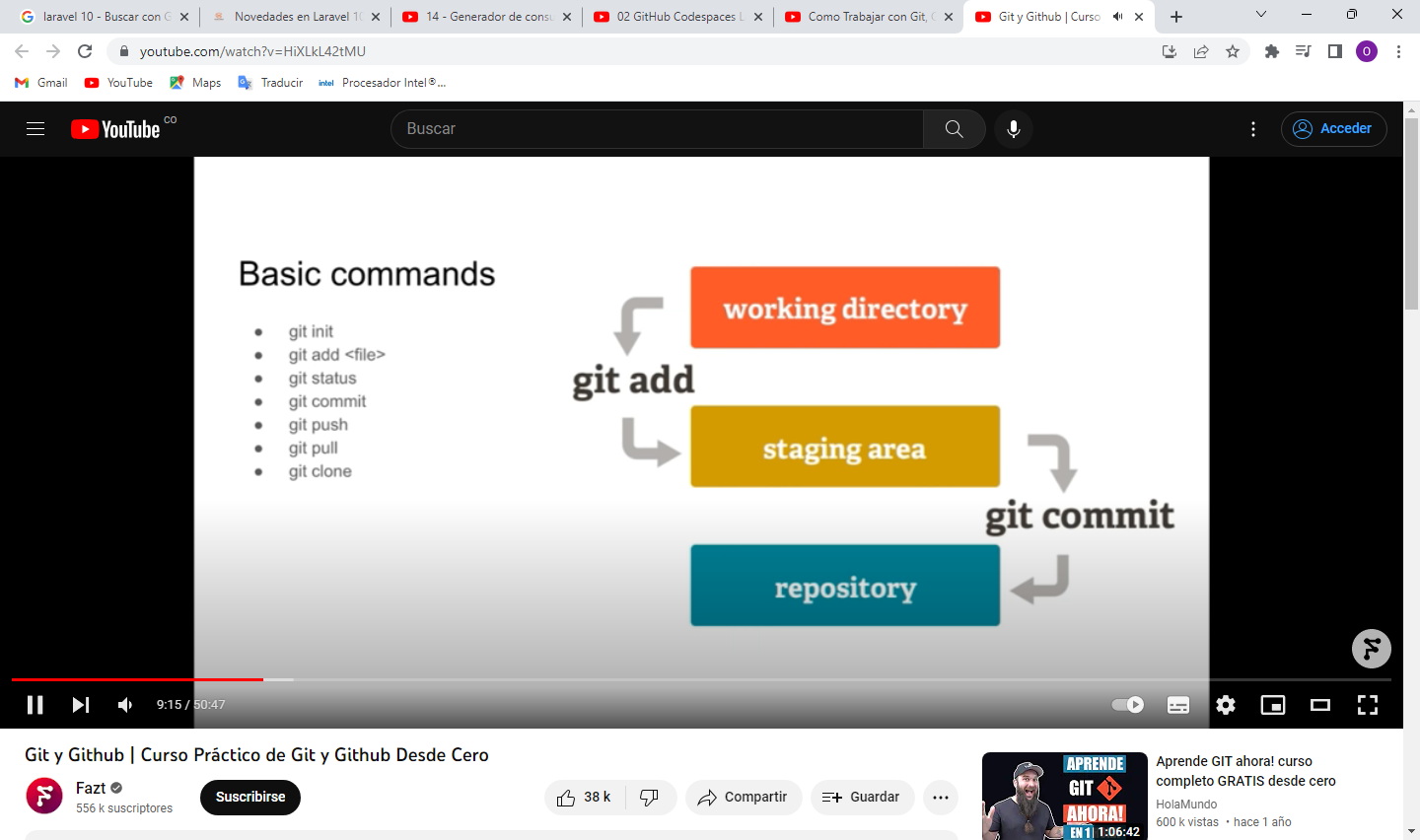
* + - * + Sincronizar. Elimina todos los roles y agrega los roles indicados.

$user->roles()->sync([1,2])

* + - Relaciones Polimórifcas
      * Se puede usar el método Create y los métodos de las relaciones muchos a muchos (Attach, Detach, Sync y Update).
      * Sumado a eso permite Crear registros directamente desde la relación de la siguiente forma: $user->image()->create(['url' => 'url 1']);
    - Demás relaciones
      * Se puede usar el método Create.
  + Notas
    - Si la relación es de muchos a muchos no se debe crear modelo de la tabla extra.
    - Las relaciones de uno a uno y de muchos a mucho se manejan desde cada modelo. La respectiva relación y la relación inversa.
    - Relación de Uno a Uno:
      * Entidad Fuerte: return $this->hasone(Profile::class);
      * Entidad Debil: return $this->belongsTo(User::class);
    - Relación Uno a Muchos:
      * Directa (Muchos): return $this->hasMany(Post::class);
      * Inversa (Uno): return $this->belongsTo(User::class);
    - Relación Muchos a Muchos:
      * En ambos modelos:
        + return $this->belongsToMany(Role::class);
        + return $this->belongsToMany(User::class);
* Uso de Tinker
  + Permite ejecutar código php en linea de comando, para por ejemplo crear registros en bd
* Seeders
  + Factories para llenar datos de prueba.
    - En el archivo Seeder
      * User::factory(10)->create();
      * Curso::factory(50)->create();
    - En el factory se definen las reglas para la generación de los datos:
* return [
* //
* 'name' => $this->faker->sentence(),
* 'descripcion' => $this->faker->paragraph(),
* 'categoria' => $this->faker->randomElement(['Desarrollo Web', 'Diseño Web', 'Idiomas'])
* ];
* Eloquent
  + Todos los registros: $cursos = Curso::all()
  + Con Filtro y ordenación: $cursos = Curso::where('categoria','Desarrollo Web')->orderby('id','desc')->get()
  + Filtro diferente a igual: $cursos = Curso::where('id','>',45)->get()
  + Búsqueda con like: $curso = Curso::where('descripcion','like','% error %')->get()
  + El primer registro de la colección: Curso::where('categoria','Desarrollo Web')->orderby('name','desc')->first()
  + $cursos = Curso::select('name','descripcion')->get()
  + Select con campos con alias $cursos = Curso::select('id as numeroId','name as title')->where('categoria','Desarrollo Web')->orderby('name','desc')->get(Idiomas)
  + Búsqueda por ID (por la llave primaria de la tabla): $cursos = Curso::find(5)
  + Paginar los resultados: $cursos = Curso::Paginate()
* Mutadores y Accesores:
  + Mutador: Cambiar lo que trae de la vista antes de guardar.
* //Mutador
* protected function name(): Attribute
* {
* return new Attribute(
* set: function($value){
* return strtolower($value);
* }
* );
* }
  + Accesor: Cambiar lo que trae la base de datos de mostrar en la vista (app). No modificar valor en base de datos, solo transforma su representación (forma de mostrarlo)
* Agregar Lenguaje / Traducción al español de los mensajes:
  + composer require laravel-lang/common --dev
  + php artisan lang:add es
  + php artisan lang:update
* Crear Form Request: Sirven para hacer las validaciones de formulario
  + php artisan make:request StoreCurso
  + Métodos
    - public function authorize(): bool => Verifica permisos de usuario para usar el componente
    - public function rules(): array => Reglas de validación
    - public function attributes(): array => Permite personalizar el nombre con el que se quiere mostrar el campo.
    - public function messages() => Personalizar mensajes de validación
* Middlewares
  + Filtra solicitudes HTTP. Por ejemplo el Middleware AUTH.
  + Los Middleware que se creen deben ser registrados en Kernel.php.
* OTROS DE INTERES
  + Personalizar Plantilla Admin LTE
    - <https://www.youtube.com/watch?v=o40pQPh-J4E>
  + Bootstrap con Laravel
    - Laravel 10 y Laravel UI Bootsrap
      * <https://techsolutionstuff.com/post/laravel-10-bootstrap-auth-scaffolding-example>
      * <https://www.kreaweb.be/laravel-10-bootstrap-5/>
    - Bootstrap con Laravel 9 y Vite:
      * <https://www.youtube.com/watch?v=D2fs-DZvFFg>
  + Agregar campo al modelo de usuarios por defecto:
    - <https://codea.app/cursos/laravel-pagina-web-administrable/9>
  + Hacer un buscador autocompletable
    - <https://www.youtube.com/watch?v=mjNT5abiYYI>
  + Enviar EMAIL:
    - Config -> Mail.php: archivo donde configuración donde se deben registrar los parámetros del servidor de correo a utilizar. Se usa también el Archio .env como en la parametrización de BD.
    - Se puede conectar a:
      * Proveedor servidor SMTP
      * Proveedor basado en API. (Trae unos por defecto)
    - Se debe crear un Mailable:
      * php artisan make:mail ContactanosMailable
      * <https://www.itsolutionstuff.com/post/laravel-10-mail-laravel-10-send-mail-tutorialexample.html>
  + Bases de Datos
    - Software para realizar un entidad-relación:
      * Diawx (Modelo lógico), Workbench (MER)



* + - Convenciones:
      * Nombres de tablas en plural.
      * En relaciones de uno a uno existen entidades fuertes y entidades débiles. Se deben crear primero las entidades fuertes.
      * Las llaves primarias deben ser nombradas con la denominación [id] y cuando son llaves foráneas debe ser [nombre tabla]\_[id]
      * Relación de muchos a muchos: El nombre debe tener el nombre de las dos tablas en singular y organizadas en orden alfabético
    - Relaciones polimórficas:
      * Se debe colocar un campo id y otro campo type. El primer campo indicará el valor de la llave primaria y el segundo indicará cual es el modelo que procesa ese registro y por ende la relación con la segunda tabla.
      * La convención es que el campo que almacenará el id, se nombre con el nombre de la tabla en singular \_ id y el campo tipo con el nombre de la tabla en singular \_ type.
      * Por ejemplo si la tabla se llama Images los campos deberían ser imagenable\_id y imagenable\_type.
      * Se modela con una llave compuesta que estará definida por los campos id y type.
    - Migraciones:
      * Crear Tabla:
        + Convención Migración: todo en minúscula y en plural. create\_[nombre tabla plural]\_table.
        + Comando Migración: php artisan make:migration create\_profiles\_table
        + Modelo: Primeras Letras en Mayúscula y en singular [Nombre Tabla]
        + Comando Modelo: php artisan make:model Profile
        + **Crea tanto Migración y Modelo: php artisan make:model Profile -m**
        + **Crea Migración, Modelo, Controlador, Seeder y Factoria: php artisan make:model Profile -mcsf o en su defecto php artisan make:model Profile -a**
    - Guards y Fillable:
      * En el almacenamiento masivo, estos elementos me permiten indicar cuales son los campos que se almacenaran (Fillable) o cuales no se almacenarán (Guards).
      * Son arrays.
      * Si se definen vacios, quiere decir que ningún campo se almacenará o no se almacenará ($guarded = []).
* GIT



* CONCEPTOS BASICOS DE GIT
  + https://www.youtube.com/watch?v=HiXLkL42tMU
  + <https://www.youtube.com/watch?v=jGehuhFhtnE>
* PUll request:
  + <https://platzi.com/clases/1557-git-github/19957-utilizando-pull-requests-en-github/>
* Git Fetch vs Git Pull:
  + <https://www.freecodecamp.org/espanol/news/git-fetch-vs-pull-cual-es-la-diferencia-entre-los-comandos-git-fetch-y-git-pull/>
* Curso de Git y Github: 7. Git como servidor y cliente
  + <https://www.youtube.com/watch?v=iWDBh4zYNcU>