



ESCUELA POLITÉCNICA NACIONAL
ESCUELA DE FORMACIÓN DE TECNÓLOGOS



DESARROLLO DE APLICACIONES WEB
(TDSD414)

ASIGNATURA:

Desarrollo de Aplicaciones Web

PROFESOR:

Ing. Ivonne Maldonado

PERÍODO ACADÉMICO:

2022-B

LABORATORIO - 9

TÍTULO:

RELACIONES POLIMORFICAS



PROPÓSITO DE LA PRÁCTICA

Familiarizar al estudiante con el manejo de relaciones polimórficas en Laravel.

OBJETIVO GENERAL

Determinar el manejo de relaciones polimórficas.

OBJETIVOS ESPECÍFICOS

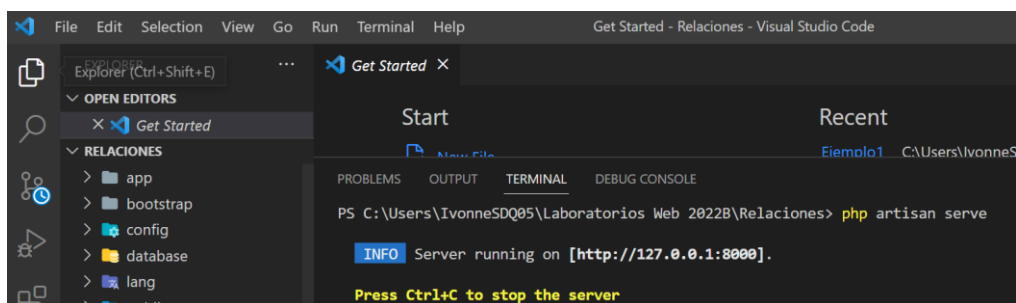
- Comprender como funciona las relaciones polimórficas.
- Comprender como visualizar los resultados de las relaciones en un proyecto.
- Visualizar los resultados obtenidos.

LINK DE GITHUB

<https://github.com/IvonneSDQ/Relaciones>

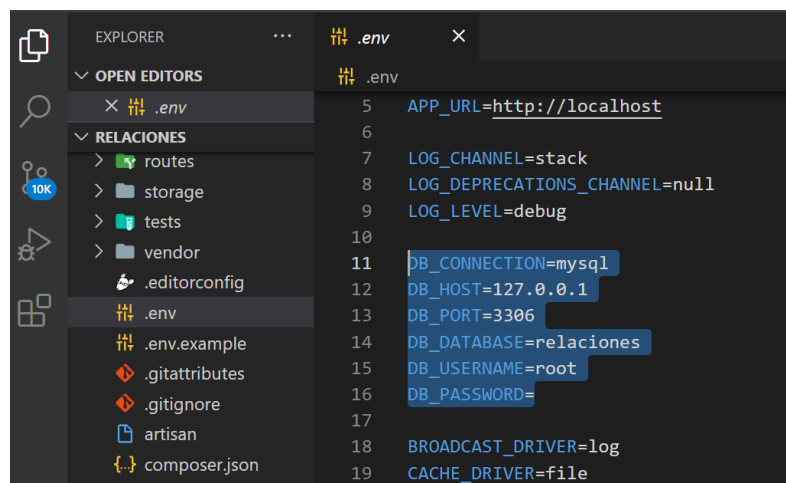
INSTRUCCIONES

1. Trabaje en el mismo proyecto de “Relaciones”



2. Compruebe la conexión con la base de datos y que cada tabla se encuentre migrada de manera correcta.

<https://laravel.com/docs/9.x/database#introduction>
<https://laravel.com/docs/9.x/migrations#running-migrations>



3. Trabajar con las relaciones polimórficas

<https://laravel.com/docs/9.x/eloquent#introduction>
<https://laravel.com/docs/9.x/eloquent-relationships#introduction>

RELACIÓN DE UNO A UNO

<https://laravel.com/docs/9.x/eloquent-relationships#one-to-one-polymorphic-relations>

ANTES

usuarios		
id	nombre	email
1	Ivonne	ivonne@gmail.com
2	Edison	edison@gmail.com
3	Gaby	gaby@hotmail.com

imagen		
id	url	id_usuario
1	imagen-ivonne	1
2	imagen-edison	2
3	imagen-gaby	3

usuarios		
id	nombre	email
1	Ivonne	ivonne@gmail.com
2	Edison	edison@gmail.com
3	Gaby	gaby@hotmail.com
1	Ivonne	ivonne@gmail.com

posts		
id	nombre	detalle
1	Post1	artículo de laravel
2	Post2	artículo de figma
3	Post3	artículo de heroku

imagen		
id	url	id_post
1	imagen-post1	1
2	imagen-post2	2
3	imagen-post3	3

posts		
id	nombre	detalle
1	Post1	artículo de laravel
2	Post2	artículo de figma
3	Post3	artículo de heroku

videos		
id	nombre	detalle
1	Video1	Video de redes
2	Video2	artículo de figma
3	Video3	artículo de heroku

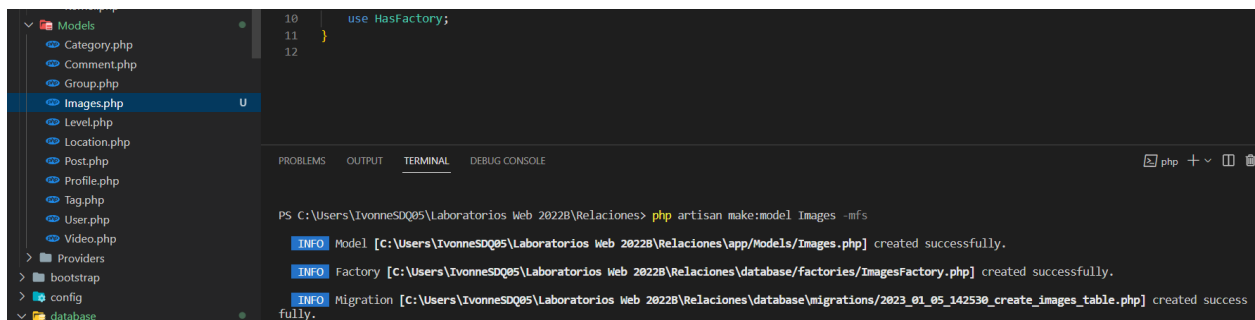
imagen		
id	url	id_video
1	imagen-video1	1
2	imagen-video2	2
3	imagen-video3	3

videos		
id	nombre	detalle
1	Video1	Video de redes
2	Video2	artículo de figma
3	Video3	artículo de heroku

images			
id	url	imageable_id	imageable_type
1	foto-1	1	App\Models\User
2	foto-1	1	App\Models\Post
3	foto-1	1	App\Models\Video

1. Crear un nuevo modelo llamado **Images**, con sus respectiva migración, factory y seeder utilizando el comando: **php artisan make:model Image -mfs**

<https://laravel.com/docs/9.x/eloquent#generating-model-classes>



2. Definir los campos (nombre de las columnas) para la tabla de la base de datos, esto se lo realiza en la migración del modelo respectivo.

NOTA: Recordar que tenemos una clave primaria compuesta

Este procedimiento realiza la relación a nivel de tablas de la **Base de Datos** únicamente.

<https://laravel.com/docs/9.x/migrations#tables>

```

10  * Run the migrations.
11
12  * @return void
13  */
14  public function up()
15  {
16      Schema::create('images', function (Blueprint $table) {
17          $table->string('url');
18          $table->unsignedBigInteger('imageable_id');
19          $table->string('imageable_type');
20          $table->primary('imageable_id', 'imageable_type');
21          $table->timestamps();
22      });
23  }

```

Si dejamos el ID y especificamos directamente la relación polimórfica.

```

7  class CreateImagesTable extends Migration
8  {
9
10     public function up()
11     {
12         Schema::create('images', function (Blueprint $table) {
13
14             $table->id();
15             $table->string('url');
16             $table->morphs('imageable');
17             $table->timestamps();
18         });
19     }
20 }

```

Código

```

$table->string('url');
$table->unsignedBigInteger('imageable_id');
$table->string('imageable_type');
$table->primary('imageable_id', 'imageable_type');
$table->timestamps();

```

3. Realizar la migración: **php artisan migrate**
4. El mismo procedimiento debe hacerse a nivel de **Eloquent de Laravel** (es decir a nivel de modelos).

<https://laravel.com/docs/9.x/eloquent-relationships#one-to-one-polymorphic-relations>

Primero en el modelo encargado de administrar la tabla Image se debe especificar una función encargada de la relación polimórfica (**morphTo**)

```

8  class Images extends Model
9  {
10     use HasFactory;
11
12     public function imageable()
13     {
14         return $this->morphTo();
15     }
16 }

```

Código

```
public function imageable()
{
    return $this->morphTo();
}
```

Ahora en los modelos Post y User (morphOne – ya que es una relación de uno a uno, además de que necesita un segundo parámetro y es el nombre del método que haya especificado en el modelo Image)

```

// RELACIÓN POLIMÓRFICA DE UNO A UNO
public function image()
{
    return $this->morphOne(Image::class, 'imageable');
}
```

```

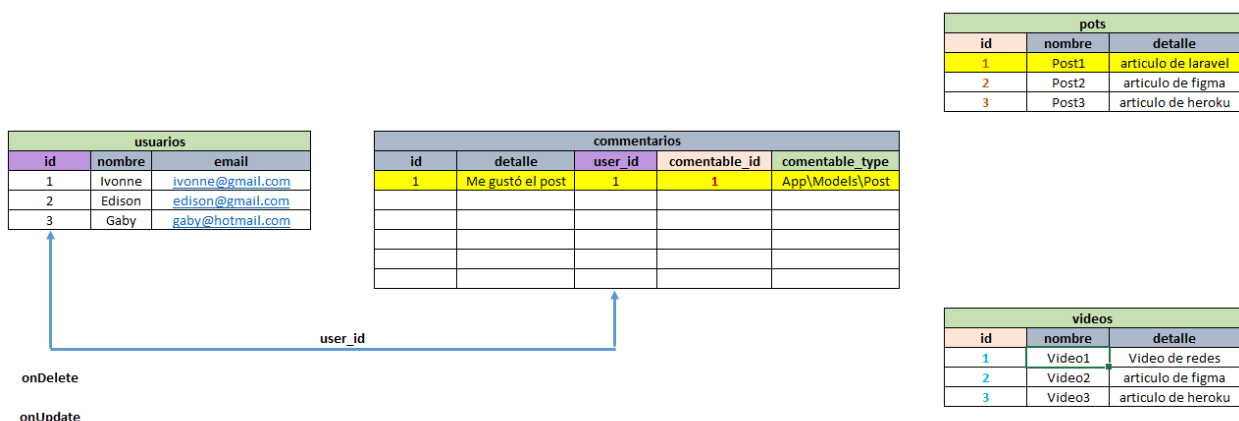
// RELACIÓN POLIMÓRFICA DE UNO A UNO
public function image()
{
    return $this->MorphOne(Image::class, 'imageable');
}
```

Código

```
// RELACIÓN POLIMÓRFICA DE UNO A UNO
public function image()
{
    return $this->MorphOne(Image::class, 'imageable');
}
```

RELACIÓN DE UNO A MUCHOS

<https://laravel.com/docs/9.x/eloquent-relationships#one-to-many-polymorphic-relations>



1. Crear un nuevo modelo llamado **Comment**, con sus respectiva migración, factory y seeder utilizando el comando: **php artisan make:model Comment -mfs**

<https://laravel.com/docs/9.x/eloquent#generating-model-classes>

2. Definir los campos (nombre de las columnas) para la tabla de la base de datos, esto se lo realiza en la migración del modelo respectivo.

Este procedimiento realiza la relación a nivel de tablas de la **Base de Datos** únicamente.

<https://laravel.com/docs/9.x/migrations#tables>

```

2021_10_12_130553_create_posts_table.php      14
2021_10_12_130614_create_videos_table.php      15
2021_10_12_195304_create_groups_table.php      16
2021_10_12_201840_create_group_user_table.php  17
2021_10_12_233859_create_comments_table.php    18
2021_10_13_011339_create_tags_table.php        19
2021_10_13_012314_create_taggables_table.php   20
2021_10_13_020048_create_locations_table.php   21
2023_01_04_144228_add_post_id_to_users.php     22
2023_01_04_145932_add_video_id_to_users.php    23
2023_01_04_150026_add_video_id_to_categories... 24
2023_01_04_150050_add_post_id_to_categories.p... 25
2023_01_04_150050_add_post_id_to_categories.p... 26
2023_01_04_150050_add_post_id_to_categories.p... 27

```

```

$table->id();
$table->text('detalle');
$table->unsignedBigInteger('user_id');
$table->string('commentable_type');
$table->unsignedBigInteger('user_id');

$table->foreign('user_id')
    ->references('id')
    ->on('users')
    ->onDelete('cascade')
    ->onUpdate('cascade');

```

De forma directa con la relación polimórfica

```

2021_10_12_130553_create_posts_table.php      14
2021_10_12_130614_create_videos_table.php      15
2021_10_12_195304_create_groups_table.php      16
2021_10_12_201840_create_group_user_table.php  17
2021_10_12_233859_create_comments_table.php    18
2021_10_13_011339_create_tags_table.php        19
2021_10_13_012314_create_taggables_table.php   20
2021_10_13_020048_create_locations_table.php   21
2023_01_04_144228_add_post_id_to_users.php     22
2023_01_04_145932_add_video_id_to_users.php    23
2023_01_04_150026_add_video_id_to_categories... 24
2023_01_04_150050_add_post_id_to_categories.p... 25
2023_01_04_150050_add_post_id_to_categories.p... 26
2023_01_05_142530_create_images_table.php      27

```

```

$table->id();
$table->text('detalle');
$table->unsignedBigInteger('user_id');
$table->morphs('commentable');

$table->foreign('user_id')
    ->references('id')
    ->on('users')
    ->onDelete('cascade')
    ->onUpdate('cascade');

$table->timestamps();
});

```

Código

```

$table->id();
    $table->text('detalle');
    $table->unsignedBigInteger('user_id');
    $table->string('commentable_type');
    $table->unsignedBigInteger('user_id');

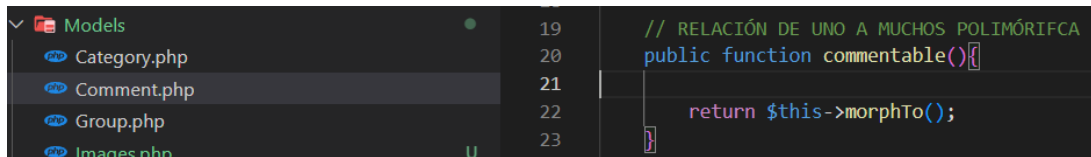
    $table->foreign('user_id')
        ->references('id')
        ->on('users')
        ->onDelete('cascade')
        ->onUpdate('cascade');

```

3. Realizar la migración: **php artisan migrate**
4. El mismo procedimiento debe hacérselo a nivel de **Eloquent de Laravel** (es decir a nivel de modelos).

<https://laravel.com/docs/9.x/eloquent-relationships#one-to-many-polymorphic-relations>

Primero en el modelo encargado de administrar la tabla Comment se debe especificar una función encargada de la relación polimórfica (**morphTo**)



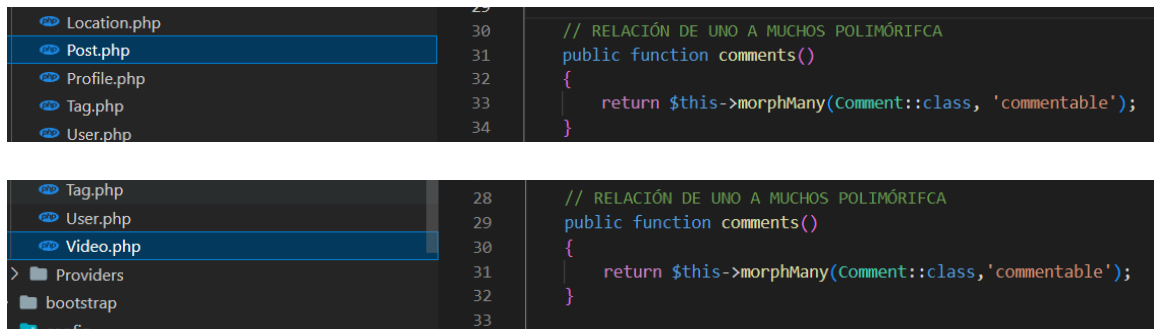
Código

```
public function commentable(){

    return $this->morphTo();

}
```

Ahora en los modelos Post y Video (morphMany – ya que es una relación de uno a muchos, además de que necesita un segundo parámetro y es el nombre del método que haya especificado en el modelo Comment)

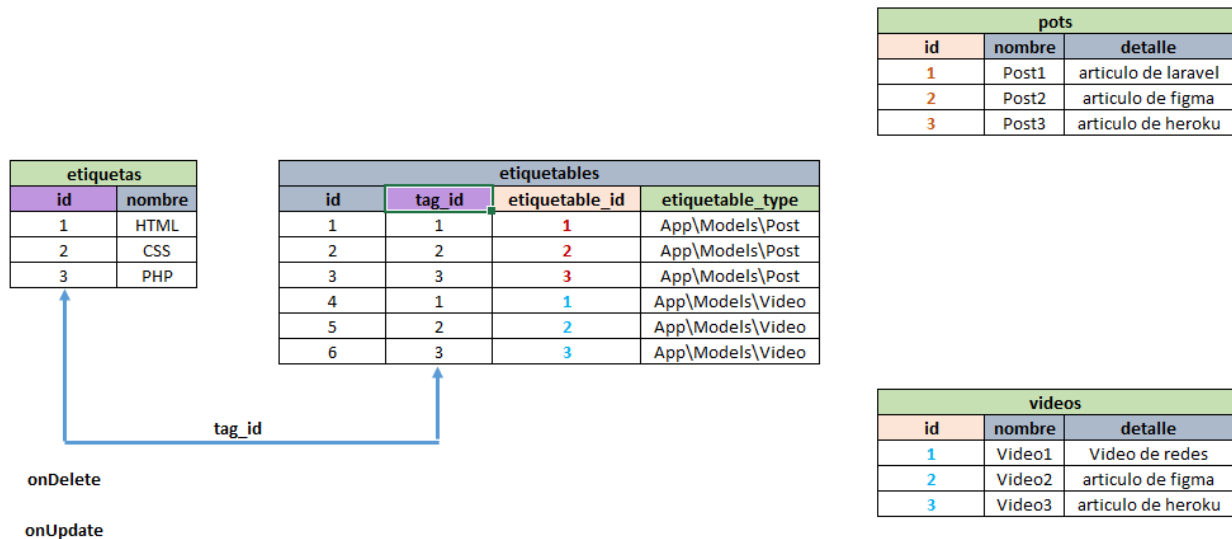


Código

```
// RELACIÓN DE UNO A MUCHOS POLIMÓRFICA
public function comments()
{
    return $this->morphMany(Comment::class, 'commentable');
}
```

RELACIÓN DE MUCHOS A MUCHOS

<https://laravel.com/docs/9.x/eloquent-relationships#many-to-many-polymorphic-relations>



1. Crear un nuevo modelo llamado **Tag**, con sus respectiva migración, factory y seedler utilizando el comando: **php artisan make:model Tag -mfs**

<https://laravel.com/docs/9.x/eloquent#generating-model-classes>

2. Definir los campos (nombre de las columnas) para la tabla de la base de datos, esto se lo realiza en la migración del modelo respectivo.

Este procedimiento realiza la relación a nivel de tablas de la **Base de Datos** únicamente.

<https://laravel.com/docs/9.x/migrations#tables>

```
2021_10_12_201840_create_group_user_table.php 13
2021_10_12_233859_create_comments_table.php 14
2021_10_13_011339_create_tags_table.php 15
2021_10_13_012314_create_taggables_table.php 16
2021_10_13_020048_create_locations_table.php 17
```

```
$table->id();
$table->string('nombre');
$table->timestamps();
});
```

Código

```
$table->id();
    $table->string('nombre');
    $table->timestamps();
```

Ahora generar la tabla intermedia: **php artisan make:migration create_taggables_table**
Y añadimos los campos

```
2021_10_12_130553_create_posts_table.php 14
2021_10_12_130614_create_videos_table.php 15
2021_10_12_195304_create_groups_table.php 16
2021_10_12_201840_create_group_user_table.php 17
2021_10_12_233859_create_comments_table.php 18
2021_10_13_011339_create_tags_table.php 19
2021_10_13_012314_create_taggables_table.php 20
2021_10_13_020048_create_locations_table.php 21
```

```
$table->id();
$table->unsignedBigInteger('tag_id');
$table->morphs('taggable');

$table->foreign('tag_id')
    ->references('id')
    ->on('tags')
    ->onDelete('cascade')
    ->onUpdate('cascade');
```


Código

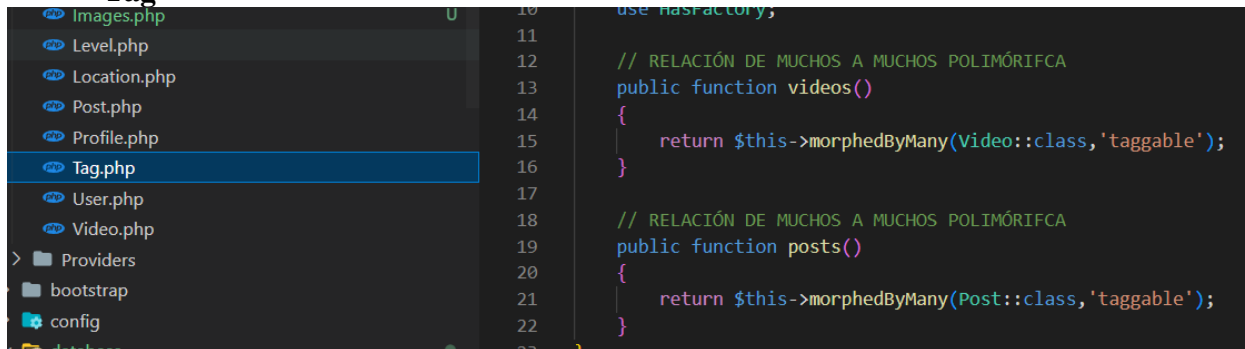
```
$table->id();
    $table->unsignedBigInteger('tag_id');
    $table->morphs('taggable');

    $table->foreign('tag_id')
        ->references('id')
        ->on('tags')
        ->onDelete('cascade')
        ->onUpdate('cascade');
```

3. Realizar la migración: **php artisan migrate**
4. El mismo procedimiento debe hacerse a nivel de **Eloquent de Laravel** (es decir a nivel de modelos).

<https://laravel.com/docs/9.x/eloquent-relationships#one-to-many-polymorphic-relations>

Tag



```
use HasFactory;

// RELACIÓN DE MUCHOS A MUCHOS POLIMÓRFICA
public function videos()
{
    return $this->morphedByMany(Video::class, 'taggable');
}

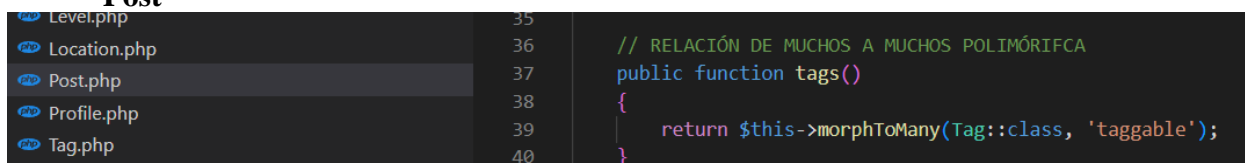
// RELACIÓN DE MUCHOS A MUCHOS POLIMÓRFICA
public function posts()
{
    return $this->morphedByMany(Post::class, 'taggable');
}
```

Código

```
// RELACIÓN DE MUCHOS A MUCHOS POLIMÓRFICA
public function videos()
{
    return $this->morphedByMany(Video::class, 'taggable');
}

// RELACIÓN DE MUCHOS A MUCHOS POLIMÓRFICA
public function posts()
{
    return $this->morphedByMany(Post::class, 'taggable');
}
```

Post



```
// RELACIÓN DE MUCHOS A MUCHOS POLIMÓRFICA
public function tags()
{
    return $this->morphToMany(Tag::class, 'taggable');
}
```

Código

```
// RELACIÓN DE MUCHOS A MUCHOS POLIMÓRFICA
```

```
public function tags()
{
    return $this->morphToMany(Tag::class, 'taggable');
}
```

Video



Código

```
// RELACIÓN DE MUCHOS A MUCHOS POLIMÓRFICA
public function tags()
{
    return $this->morphToMany(Tag::class, 'taggable');
}
```

PRESENTACIÓN

Al finalizar tu tarea deberás subir:

- Un archivo en formato pdf con el nombre (Laboratorio9_ AWeb_2022B _NApellido)

RECURSOS NECESARIOS

- Visual Studio Code
- Herramienta ofimática
- Internet
- Material de clase