# 1. Preparación del proyecto.

## 1.1 Creación:

composer create-project laravel/laravel:^10 nombredelproyecto
cd nombredelproyecto
composer require laravel/breeze --dev
php artisan breeze:install
npm install -D flowbite

## 1.2 Editar archivo .env:

```
DB_CONNECTION=pgsql
DB_HOST=127.0.0.1
DB_PORT=5432
DB_DATABASE=nombrebasededatos
DB_USERNAME=nombreusuario
```

```
DB_PASSWORD=contraseña
```

### 1.3 Crear la base de datos:

sudo -u postgres createuser -P nombreusuario
sudo -u postgres createdb -O nombrebasededatos usuario

## 2. Hacer los modelos

### 2.1 Creación y edición de modelos

php artisan make:model nombre -mcr
si se cambia el nombre de una tabla en el modelo se añade lo siguiente:
protected $table = 'nombrenuevo';

#### 2.1.1 Relación muchos a muchos

Se crea la función en el modelo como siempre , en este caso ambos modelos llevaran un belongsToMany (Modelo Álbum)

```
public function artistas()
    {
        return $this->belongsToMany(Artista::class,
'album_artista');
    }
```

Se crea una tabla pivote, si la relación de muchos a muchos es entre álbumes y artistas se crea la tabla con php artisan make:migration create_album_artista_table el nombre de la tabla se pone en orden alfabético. album va antes que artista

la tabla album_artista estará compuesta por los foreingid de cada tabla original a la que pertenece junto con un primary que será el id de dicha tabla

```
$table->foreignId('album_id')->constrained('albumes');
$table->foreignId('tema_id')->constrained('temas');
$table->primary('album_id','tema_id');
```

#### 2.1.2 Relación una a muchos

```
public function proyecciones() {
        return $this->hasMany(Proyeccion::class);
    }

public function pelicula() {
      return $this->belongsTo(Pelicula::class);
}
```

hasMany: se utiliza cuando una relacion tiene muchas con respecto a la otra tabla por ejemplo pelicula tiene muchas proyecciones pero una proyeccion seria de una pelicula por lo que proyecciones usaria hasmany y pelicula en la tabla Proyección usaria belongsTo

HasManyThrough : una pelicula por ejemplo puede tener muchas entradas, pero no existe relacion directa. Una pelicula tiene muchas proyecciones y una proyeccion muchas entradas por lo que existe relación entre entradas y peliculas a traves de proyecciones. Para ello en el modelo pelicula pondremos

```
public function entradas()
    {
        return $this->hasManyThrough(Entrada::class, Proyeccion::class);
    }
```

### 2.1.3 Relación Polimórfica

```
POLIMÓRFICAS :

te diría que una polimorfica esta formada por 3 modelos, en ese
taco aula-----ordenador
                                                                  |
                                                                  |
como te dice lo de colocable (examen-recuperacion)
                                                ^
                                                                dispositivo

public function dispositivos()
    {
        return $this->morphMany(Dispositivo::class, 'colocable');
---->en aula
    }


public function dispositivos()
    {
        return $this->morphMany(Dispositivo::class, 'colocable');
------->en ordenador
    }



public function colocable()
    {
        return $this->morphTo();       -------->en dispositivo
    }
```

```
morphmany seria el equivalente a hasMany en version polomorfica y
morphTo seria belongsTo en las relaciones uno a muchos
en la migracion se le pone morphs mas el nombre que le queramos
poner
 $table->morphs('colocable');

ORIGEN Y DESTINO:

Cuando nos diga origen_id y destino_id ya sabremos que será un
aeropuerto o un aula de destino y origen,


public function cambioOrigen()
    {
        return $this->hasMany(Cambio::class, 'origen_id');
---> en aula (la que tiene el 1)
    }


public function cambioDestino()
    {
        return $this->hasMany(Cambio::class, 'destino_id');
------->en aula también
    }




public function origen()
    {
        return $this->belongsTo(Aula::class, 'origen_id');
------>en cambio
    }

    public function destino()
    {
        return $this->belongsTo(Aula::class, 'destino_id');
------>en cambio
    }
```

## Asociación en relaciones polimórficas

Vamos al php artisan tinker
creamos lo que vayamos a usar, por ejemplo un aula

\App\Models\Aula::Create([‘nombre’ => ‘P7’]);

ponemos el modelo que vamos a usar y lo asociamos

use App\Models\Aula
$a = Aula::find(numero del id que queramos);

Hacemos lo mismo con el modelo que queremos asociar al aula

use App\Models\Dispositivo
$d = new Dispositivo();
$d ->nombre = ‘Teclado’;
$d->colocable()->associate($a);
$d->save();

### 2.1.4 Relación uno a uno

```
Un telefono corresponde a un usuario

public function telefono()
    {
        return $this->hasOne(Telefono::class);
    }



public function user()
    {
        return $this->belongsTo(User::class);
    }
```

## 2.2 Edición de la base de datos

si una tabla usa la clave principal de otra tabla como campo se hará de la siguiente manera

```
$table->foreingId('pelicula_id')->constrained(‘pelicula’);
```

foreignid seguido de la tabla mas su clave principal y con el atributo constrained()
para numeros se usa integer
para palabras string
para fechas timestamp
para precios $table->decimal('precio', 8, 2);
para una longitud determinada $table->string('codigo')->unique()->length(13) o $table->string('codigo', 3);;
si queremos que se parezca a un patron  concreto :

```
DB::statement("ALTER TABLE aeropuertos
                ADD CONSTRAINT ck_codigo_letras
                CHECK (codigo SIMILAR TO '%[A-Z]{3}%')");
  }
```

para una cantidad no negativa $table->integer('cantidad')->unsigned();
si se crea una restriccion multiple se hace de la siguiente manera

```
$table->unique(['pelicula_id', 'fecha_hora', 'sala_id']);
```

INSERTAR VALORES

INSERT INTO nombretabla (campos) VALUES (‘valores’)
ELIMINAR
delete from prestamos where ejemplar_id='1' and cliente_id='1';

CON TINKER

\App\Models\nombremodelo::create([‘campo’ => ‘nombre’, ‘campo2’ => ‘nombre2’])

previamente en el modelo poner un protected fillable con los nombres de los campos

protected $fillable =  [‘campo’, ‘campo2’];

## 3.CRUD

## 3.1 Hacer y editar las vistas

lo primero que debemos hacer es cambiar la vista, para ello vamos a routes/web y ponemos la dirección de la ruta que queremos. Ejemplo:

```
Route::resource('peliculas', PeliculaController::class);
Route::resource('ordenadores',
OrdenadorController::class)->parameters(['ordenadores' =>
'ordenador']);
```

a partir de ese momento ya existirá un /películas

para hacer la vista de películas nos vamos a

resources/views y creamos la carpeta películas

copiamos los archivos, create, edit, index y show y editamos los campos
y layouts

nos puede salir el siguiente error en resources/views/layouts navigation.blade.php
**cuando se un fallo : Attempt to read property "name" on null**

se arregla introduciendolo en :

```
@auth
                {{ Auth::user()->name }}
@endauth
```

SHOW DIFERENTE

Si en lugar de que el show sea nombredelatabla/id queremos que sea
nombredelatabla/ejemplo/id

deberiamos crear primero la ruta

```
Route::get('/alumnos/criterios/{alumno}', [AlumnoController::class, 'criterios'])
   ->name('alumnos.criterios');
```

el show en vez de llamarse show se llamara criterios.blade.php

En el controlador el show estará vacio y a su vez habrá una funcion llamada criterios que devuelva la vista nueva

```
 public function criterios(Alumno $alumno)
   {
       return view('alumnos.criterios', [
           'alumno' => $alumno,
       ]);
   }
```

IMPORTANTE cambiar la ruta nueva en el archivo index.blade.php

```
Route::get('/artistas/albumes/{artista}', function (Artista $artista) {
       return view('artistas.albumes', [       //ruta de jose raposo
           'artista' => $artista,
       ]);
   })->name('artistas.albumes');
});
```

Creamos un archivo llamado albumes.blade.php dentro de la carpeta artistas

para crear una ruta específica se puede hacer en web o creando una funcion en el controlador
metodo del controlador:
public function criterios(Alumno $alumno){

}
ejemplo web:
Route::get(‘alumnos/criterios/{alumnos}, [AlumnoController::class, criterios(nombre del metodo creado en el controlador])

## 3.2 Editar el CRUD

no borrar algo según una condición

```
public function destroy(Album $album)
    {
        if (!$album->artistas->isEmpty() ||
!$album->canciones->isEmpty()) {
            echo ("no se puede borrar");
        } else {
            $album->delete();
        }
        return redirect()->route('albums.index');
    }
```

## 4.CREAR

## 4.1CREAR UNA LISTA DESPLEGABLE CON COSAS YA EXISTENTES

<x-input-label for="desarrolladora_id" :value="'desarrolladora del videojuego'" />
<select id="desarrolladora_id"
class="border-gray-300 focus:border-indigo-500 focus:ring-indigo-500 rounded-md shadow-sm block mt-1 w-full"
name="desarrolladora_id" required>
@foreach ($desarrolladoras as $desarrolladora)
<option value="{{ $desarrolladora->id }}"
{{ old('desarrolladora_id') == $desarrolladora->id ? 'selected' : '' }}
>

```
                {{ $desarrolladora->nombre }}
            </option>
          @endforeach
        </select>
        <x-input-error :messages="$errors->get('desarrolladora_id')" class="mt-2" />
      </div>
```

Si por ejemplo al crear un videojuego tenemos que poner a que desarrolladora pertenece con esto se nos aparecerá una lista desplegable con todas las desarrolladoras ya existentes. en lugar de introducir el id a mano.

## 4.2HACER UNA MEDIA

```php
public function notas() {
        return $this->HasMany(Nota::class);
    }

    public function notas_por_criterios()
    {
        return $this->notas()
            ->select(DB::raw('alumno_id, ccee_id, max(nota) AS nota'))
            ->groupBy(['alumno_id', 'ccee_id'])
            ->get();
    }
```

Si tenemos varias notas de alumnos almacenadas la media se haria con un avg(‘nombrecampo’).
Ejemplo :

```
 {{  ($alumno->notas_por_criterios()->avg('nota'))  }}
```

## 4.3CREAR FORMATO INTERVAL

```
public function up(): void
    {

        Grammar::macro('typeInterval', function () {
            return 'interval';
        });

se añade eso  justo antes del Schema….
```

```
El formato debe ser de la siguiente forma:
$table->addColumn('interval', 'duracion');

El grammar que hay que añadir es el siguiente:
use Illuminate\Database\Schema\Grammars\Grammar;
```

Insertar en formato PMT P3M20S = 3 minutos y 20 segundos
P4Y2M3DT3H2M5S = 4 años, 2 meses, 3 dias, 3 horas , 2 minutos y 5 segundos

## 4.4PODER ORDENAR POR FILAS

Paso 1 el index debe quedar tal que asi :

```
public function index(Request $request)
  {
     $order = $request->query('order', 'denominacion');
     $order_dir = $request->query('order_dir', 'asc');
     $articulos = Articulo::with(['iva', 'categoria'])
        ->selectRaw('articulos.*')
        ->leftJoin('categorias', 'articulos.categoria_id', '=', 'categorias.id')
        ->leftJoin('ivas', 'articulos.iva_id', '=', 'ivas.id')
        ->orderBy($order, $order_dir)
        ->paginate(3);
     return view('articulos.index', [
        'articulos' => $articulos,
        'order' => $order,
        'order_dir' => $order_dir,
     ]);
  }
```

con la consulta personalizada.

Crear un archivo helpers.php en app y pegar las funciones :

```
<?php

function order_dir_arrow($order, $order_dir)
{
   return $order == false ? '' : ($order_dir == 'desc' ? '↑' : '↓');
}

function order_dir($order, $order_dir)
{
   return $order == false ? 'asc' : ($order_dir == 'asc' ? 'desc' :
'asc');
}
```

Para que laravel cargue el helper agregar en el composer.json :

```
  "autoload": {
      "psr-4": {
         "App\\": "app/",
         "Database\\Factories\\": "database/factories/",
         "Database\\Seeders\\": "database/seeders/"
      },
      "files": ["app/helpers.php"]
   },
```

sustituimos ese por el que ya tenemos

despues de modificar ponemos en el terminal : composer dump-autoload

## 4.5CREAR ALGO DIRECTAMENTE AL USUARIO REGISTRADO SIN PEDIRLO EN EL FORMULARIO:

```
public function store(Request $request)
        {

            $reserva = new Reserva();
            $reserva->user_id = auth()->id();
            $reserva->vuelo_id = $request->input('vuelo_id');
            $reserva->asiento = $request->input('asiento');
            $reserva->save();

            session()->flash('success', 'cancion creado
correctamente.');
            return redirect()->route('reservas.index');
        }
```

En el store indicamos que el user_id = auth()->id();

## 5.ESTRUCTURAS

### 5.1if ( $loquesea ) { accion } else { accion 2 }

```
if ($cancion->albumes->isEmpty() and $cancion->artistas->isEmpty()) {
            $cancion->delete();
        } else {
            session()->flash('error', 'La categoría tiene artículos.');
        }
```

5.2@FOREACH ($ejemplos as $ejemplo)
@ENDFOREACH

```
<tbody>
                @foreach ($vuelos as $vuelo)
                <tr class="bg-white border-b dark:bg-gray-800
dark:border-gray-700">

                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->codigo}}
                    </th>
                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->aeropuerto_origen->nombre }}
                    </th>
                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->aeropuerto_destino->nombre}}
                    </th>
                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->precio}}
                    </th>
                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->plazas}}
                    </th>
                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->plazasDisponibles()}}
                    </th>
                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->salida}}
                    </th>
                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->llegada}}
                    </th>
                    </tr>
                @endforeach
            </tbody>
```

## 6. Errores

SQLSTATE[23503]: Foreign key violation: 7 ERROR: update o delete en «temas» viola la llave foránea «artistas_temas_tema_id_foreign» en la tabla «artistas_temas» DETAIL: La llave (id)=(3) todavía es referida desde la tabla «artistas_temas». DELETE FROM "temas" WHERE "id" = 3

Queremos borrar algo que es clave primaria de una tabla pivote. Para ello debemos borrar primero la tabla pivote.

```
public function destroy(Tema $tema)
   {

  // Eliminar las referencias en la tabla pivote
  DB::table('artistas_temas')->where('tema_id', $tema->id)->delete();

  // Luego eliminar el tema
  $tema->delete();

  // Crear un mensaje de éxito y redirigir
  session()->flash('success', 'Tema eliminado correctamente.');
  return redirect()->route('temas.index');
   }
```

## 7. Funciones

```
 public function nombres_alumnos()
   {
      $registros = Nota::where('asignatura_id', $this->id)->get();
      $nombres_alumnos = [];
      foreach ($registros as $registro) {
         $cod = $registro->alumno_id;
         $n = Alumno::find($cod);

         if (!in_array($n->nombre, $nombres_alumnos)) {
               $nombres_alumnos[] = $n->nombre;
            }

      }
      // Construir la lista de nombres de álbumes
      $lista_nombres_alumnos = '';
      foreach ($nombres_alumnos as $r) {
```

```
            $lista_nombres_alumnos .= '<li>' . $r . '</li>';
        }
        return $lista_nombres_alumnos ? '<ul>' . $lista_nombres_alumnos . '</ul>' : 'Sin
alumnos';
    }


    public function notas_primer_trimestre()
    {
        $canciones = Nota::where('asignatura_id', $this->id)
                        ->where('trimestre', '1')
                        ->get();

        $nombres_canciones = '';
        foreach ($canciones as $cancion) {
            $nombre = $cancion->nota;
            $nombres_canciones .= '<li>' . $nombre . '</li>';
        }
        return $nombres_canciones ? '<ul>' . $nombres_canciones . '</ul>' : 'Sin calificar';
    }

    public function notas_segun_trimestre()
    {
        $canciones = Nota::where('asignatura_id', $this->id)
                        ->where('trimestre', '1')
                        ->get();

        $nombres_canciones = '';
        foreach ($canciones as $cancion) {
            $nombre = $cancion->nota;
            $nombres_canciones .= '<li>' . $nombre . '</li>';
        }
        return $nombres_canciones ? '<ul>' . $nombres_canciones . '</ul>' : 'Sin calificar';
    }

  public function nombres_canciones()
    {
        $canciones = AlbumTema::where('album_id', $this->id)->get();
        $nombres_canciones = '';
        foreach ($canciones as $cancion) {
            $nombre = $cancion->tema_id;

            $al = Tema::find($nombre);
            $nombres_canciones .= '<li>' . $al->titulo . '</li>';
        }
        return $nombres_canciones ? '<ul>' . $nombres_canciones . '</ul>' : 'Sin canciones';
    }
```

```
    public function duracion_album()
    {
        $registros = AlbumTema::where('album_id', $this->id)->get();
        $duracion = 0;

        foreach ($registros as $registro) {
            $cancion = Tema::find($registro->tema_id);
            $tiempo = $cancion->duracion;

            $total_cancion = Carbon::createFromFormat('H:i:s', $tiempo);
            $duracion += $total_cancion->hour * 3600 + $total_cancion->minute * 60 +
$total_cancion->second;
        }

        // Convertir la duración total de segundos a formato minutos:segundos
        $minutos = floor($duracion / 60);
        $segundos = $duracion % 60;

        return sprintf('%02d:%02d', $minutos, $segundos);
    }
```

## 8. Gates Y Politicas

### 1. Definir Gates

**Las gates se definen en el `AuthServiceProvider`. Aquí es donde se registra toda la lógica de autorización.**

**Ejemplo de definición de una gate:**

```
use Illuminate\Support\Facades\Gate;

class AuthServiceProvider extends ServiceProvider
{
    public function boot()
    {
        $this->registerPolicies();

        Gate::define('update-post', function ($user,
$post) {
```

```
            return $user->id === $post->user_id;
        });
    }
}
```

**En este ejemplo, se define una gate llamada `update-post` que verifica si el ID del usuario coincide con el ID del usuario que creó el post.**

**2. Utilizar Gates en los Controladores o Vistas**

**Una vez definida una gate, puedes utilizarla en tus controladores, vistas o cualquier otro lugar de tu aplicación donde necesites verificar la autorización.**

**Ejemplo en un controlador:**

```
public function update(Request $request, Post $post)
{
    if (Gate::allows('update-post', $post)) {
        // El usuario está autorizado a actualizar el
post
        // Lógica para actualizar el post
    } else {
        // El usuario no está autorizado
        abort(403);
    }
}
```

**En este ejemplo, se utiliza `Gate::allows` para verificar si el usuario está autorizado para actualizar el post.**

**3. Métodos de Verificación**

**Laravel proporciona varios métodos para trabajar con gates:**

- `Gate::allows('gate-name', $arguments)`: Retorna `true` si el usuario está autorizado.
- `Gate::denies('gate-name', $arguments)`: Retorna `true` si el usuario no está autorizado.
- `Gate::check('gate-name', $arguments)`: Similar a `allows`, pero más semántico.
- `Gate::forUser($user)->allows('gate-name', $arguments)`: Verifica autorización para un usuario específico.

**4. Definir Políticas**

**Para manejar la lógica de autorización de manera más estructurada, Laravel proporciona las políticas, que son clases dedicadas a definir la autorización para un modelo específico.**

**Generar una política:**

```
php artisan make:policy PostPolicy
```

**Definir la política:**

```
class PostPolicy
{
    public function update(User $user, Post $post)
    {
        return $user->id === $post->user_id;
    }
}
```

**Registrar la política:**

**En el `AuthServiceProvider`:**

```
protected $policies = [
    'App\Models\Post' => 'App\Policies\PostPolicy',
];

public function boot()
{
    $this->registerPolicies();
}
```

#### Usar la política:

```
if ($user->can('update', $post)) {
    // El usuario está autorizado
}
```

#### 1. Verificación de Roles

**Puedes utilizar gates para verificar si un usuario tiene un rol específico.**

```
Gate::define('admin-access', function ($user) {
    return $user->role === 'admin';
});
```

#### Uso en un Controlador

```
public function adminDashboard()
{
    if (Gate::denies('admin-access')) {
        abort(403);
    }
```

```
    // Mostrar el panel de administración
}
```

#### 2. Permiso para Crear Posts

**Verificar si un usuario tiene permiso para crear posts.**

```
Gate::define('create-post', function ($user) {
    return $user->hasPermission('create-post');
});
```

#### Uso en un Controlador

```
public function create()
{
    if (Gate::denies('create-post')) {
        abort(403);
    }

    // Mostrar formulario de creación de post
}
```

### Usando Gates en Vistas Blade

**También puedes utilizar gates directamente en tus vistas Blade para mostrar u ocultar contenido basado en permisos.**

```
@can('update-post', $post)
    <a href="{{ route('posts.edit', $post) }}">Editar
Post</a>
@endcan
```

@cannot

# 1. Preparación del proyecto.

## 1.1 Creación:

composer create-project laravel/laravel:^10 nombredelproyecto
cd nombredelproyecto
composer require laravel/breeze --dev
php artisan breeze:install
npm install -D flowbite

## 1.2 Editar archivo .env:

```
DB_CONNECTION=pgsql
DB_HOST=127.0.0.1
DB_PORT=5432
DB_DATABASE=nombrebasededatos
DB_USERNAME=nombreusuario
DB_PASSWORD=contraseña
```

### 1.3 Crear la base de datos:

sudo -u postgres createuser -P nombreusuario
sudo -u postgres createdb -O nombrebasededatos usuario

## 2. Hacer los modelos

### 2.1 Creación y edición de modelos

php artisan make:model nombre -mcr
si se cambia el nombre de una tabla en el modelo se añade lo siguiente:
protected $table = 'nombrenuevo';

#### 2.1.1 Relación muchos a muchos

Se crea la función en el modelo como siempre , en este caso ambos modelos llevaran un belongsToMany (Modelo Álbum)

```
public function artistas()
    {
        return $this->belongsToMany(Artista::class,
'album_artista');
    }
```

Se crea una tabla pivote, si la relación de muchos a muchos es entre álbumes y artistas se crea la tabla con php artisan make:migration create_album_artista_table el nombre de la tabla se pone en orden alfabético. album va antes que artista

la tabla album_artista estará compuesta por los foreingid de cada tabla original a la que pertenece junto con un primary que será el id de dicha tabla

```
$table->foreignId('album_id')->constrained('albumes');
$table->foreignId('tema_id')->constrained('temas');
$table->primary('album_id','tema_id');
```

#### 2.1.2 Relación una a muchos

```
public function proyecciones() {
        return $this->hasMany(Proyeccion::class);
    }

public function pelicula() {
      return $this->belongsTo(Pelicula::class);
}
```

hasMany: se utiliza cuando una relacion tiene muchas con respecto a la otra tabla por ejemplo pelicula tiene muchas proyecciones pero una proyeccion seria de una pelicula por lo que proyecciones usaria hasmany y pelicula en la tabla Proyección usaria belongsTo

HasManyThrough : una pelicula por ejemplo puede tener muchas entradas, pero no existe relacion directa. Una pelicula tiene muchas proyecciones y una proyeccion muchas entradas por lo que existe relación entre entradas y peliculas a traves de proyecciones. Para ello en el modelo pelicula pondremos

```php
public function entradas()
    {
        return $this->hasManyThrough(Entrada::class, Proyeccion::class);
    }
```

### 2.1.3 Relación Polimórfica

```
POLIMÓRFICAS :

te diría que una polimorfica esta formada por 3 modelos, en ese
taco aula-----ordenador
                                                                |
                                                                |
como te dice lo de colocable (examen-recuperacion)
                                                ^
                                                                dispositivo

public function dispositivos()
    {
        return $this->morphMany(Dispositivo::class, 'colocable');
---->en aula
    }


public function dispositivos()
    {
        return $this->morphMany(Dispositivo::class, 'colocable');
------->en ordenador
    }



public function colocable()
    {
        return $this->morphTo();        -------->en dispositivo
    }
```

```
morphmany seria el equivalente a hasMany en version polomorfica y
morphTo seria belongsTo en las relaciones uno a muchos
en la migracion se le pone morphs mas el nombre que le queramos
poner
 $table->morphs('colocable');

ORIGEN Y DESTINO:

Cuando nos diga origen_id y destino_id ya sabremos que será un
aeropuerto o un aula de destino y origen,


public function cambioOrigen()
    {
        return $this->hasMany(Cambio::class, 'origen_id');
---> en aula (la que tiene el 1)
    }


public function cambioDestino()
    {
        return $this->hasMany(Cambio::class, 'destino_id');
------->en aula también
    }




public function origen()
    {
        return $this->belongsTo(Aula::class, 'origen_id');
------>en cambio
    }

    public function destino()
    {
        return $this->belongsTo(Aula::class, 'destino_id');
------>en cambio
    }
```

## Asociación en relaciones polimórficas

Vamos al php artisan tinker
creamos lo que vayamos a usar, por ejemplo un aula
\App\Models\Aula::Create([‘nombre’ => ‘P7’]);

ponemos el modelo que vamos a usar y lo asociamos

use App\Models\Aula
$a = Aula::find(numero del id que queramos);

Hacemos lo mismo con el modelo que queremos asociar al aula

use App\Models\Dispositivo
$d = new Dispositivo();
$d ->nombre = ‘Teclado’;
$d->colocable()->associate($a);
$d->save();

### 2.1.4 Relación uno a uno

```
Un telefono corresponde a un usuario

public function telefono()
    {
        return $this->hasOne(Telefono::class);
    }



public function user()
    {
        return $this->belongsTo(User::class);
    }
```

## 2.2 Edición de la base de datos

si una tabla usa la clave principal de otra tabla como campo se hará de la siguiente manera

```
$table->foreingId('pelicula_id')->constrained(‘pelicula’);
```

foreignid seguido de la tabla mas su clave principal y con el atributo constrained()
para numeros se usa integer
para palabras string
para fechas timestamp
para precios $table->decimal('precio', 8, 2);
para una longitud determinada $table->string('codigo')->unique()->length(13) o $table->string('codigo', 3);;
si queremos que se parezca a un patron  concreto :

DB::statement("ALTER TABLE aeropuertos

```
            ADD CONSTRAINT ck_codigo_letras
            CHECK (codigo SIMILAR TO '%[A-Z]{3}%')");
    }
```

para una cantidad no negativa $table->integer('cantidad')->unsigned();
si se crea una restriccion multiple se hace de la siguiente manera

```
$table->unique(['pelicula_id', 'fecha_hora', 'sala_id']);
```

INSERTAR VALORES

INSERT INTO nombretabla (campos) VALUES (‘valores’)
ELIMINAR
delete from prestamos where ejemplar_id='1' and cliente_id='1';

CON TINKER

\App\Models\nombremodelo::create([‘campo’ => ‘nombre’, ‘campo2’ => ‘nombre2’])

previamente en el modelo poner un protected fillable con los nombres de los campos

protected $fillable =  [‘campo’, ‘campo2’];

## 3.CRUD

## 3.1 Hacer y editar las vistas

lo primero que debemos hacer es cambiar la vista, para ello vamos a routes/web y ponemos la dirección de la ruta que queremos. Ejemplo:

```
Route::resource('peliculas', PeliculaController::class);
Route::resource('ordenadores',
OrdenadorController::class)->parameters([‘ordenadores’ =>
‘ordenador’]);
```

a partir de ese momento ya existirá un /películas

para hacer la vista de películas nos vamos a

resources/views y creamos la carpeta películas

copiamos los archivos, create, edit, index y show y editamos los campos
y layouts

nos puede salir el siguiente error en resources/views/layouts navigation.blade.php
**cuando se un fallo : Attempt to read property "name" on null**

se arregla introduciendolo en :

```
@auth
                {{ Auth::user()->name }}
@endauth
```

SHOW DIFERENTE

Si en lugar de que el show sea nombredelatabla/id queremos que sea
nombredelatabla/ejemplo/id

deberiamos crear primero la ruta

```
Route::get('/alumnos/criterios/{alumno}', [AlumnoController::class, 'criterios'])
    ->name('alumnos.criterios');
```

el show en vez de llamarse show se llamara criterios.blade.php

En el controlador el show estará vacio y a su vez habrá una funcion llamada criterios que
devuelva la vista nueva

```
 public function criterios(Alumno $alumno)
    {
        return view('alumnos.criterios', [
            'alumno' => $alumno,
        ]);
    }
```

IMPORTANTE cambiar la ruta nueva en el archivo index.blade.php

```
Route::get('/artistas/albumes/{artista}', function (Artista $artista) {
        return view('artistas.albumes', [       //ruta de jose raposo
            'artista' => $artista,
        ]);
    })->name('artistas.albumes');
});
```

Creamos un archivo llamado albumes.blade.php dentro de la carpeta artistas

para crear una ruta específica se puede hacer en web o creando una funcion en el controlador
metodo del controlador:
public function criterios(Alumno $alumno){

}
ejemplo web:
Route::get(‘alumnos/criterios/{alumnos}, [AlumnoController::class, criterios(nombre del metodo creado en el controlador])

## 3.2 Editar el CRUD

no borrar algo según una condición

```
public function destroy(Album $album)
    {
        if (!$album->artistas->isEmpty() ||
!$album->canciones->isEmpty()) {
            echo ("no se puede borrar");
        } else {
            $album->delete();
        }
        return redirect()->route('albums.index');
    }
```

## 4.CREAR

## 4.1CREAR UNA LISTA DESPLEGABLE CON COSAS YA EXISTENTES

```
<div class="mt-4">
            <x-input-label for="desarrolladora_id" :value="'desarrolladora del videojuego'"
/>
            <select id="desarrolladora_id"
                class="border-gray-300 focus:border-indigo-500 focus:ring-indigo-500
rounded-md shadow-sm block mt-1 w-full"
                name="desarrolladora_id" required>
                @foreach ($desarrolladoras as $desarrolladora)
                    <option value="{{ $desarrolladora->id }}"
                        {{ old('desarrolladora_id') == $desarrolladora->id ? 'selected' : '' }}
                        >
                        {{ $desarrolladora->nombre }}
```

```
            </option>
          @endforeach
        </select>
        <x-input-error :messages="$errors->get('desarrolladora_id')" class="mt-2" />
      </div>
```

Si por ejemplo al crear un videojuego tenemos que poner a que desarrolladora pertenece con esto se nos aparecerá una lista desplegable con todas las desarrolladoras ya existentes. en lugar de introducir el id a mano.

## 4.2HACER UNA MEDIA

```
public function notas() {
        return $this->HasMany(Nota::class);
    }

    public function notas_por_criterios()
    {
        return $this->notas()
            ->select(DB::raw('alumno_id, ccee_id, max(nota) AS nota'))
            ->groupBy(['alumno_id', 'ccee_id'])
            ->get();
    }
```

Si tenemos varias notas de alumnos almacenadas la media se haria con un avg(‘nombrecampo’).
Ejemplo :

```
 {{ ($alumno->notas_por_criterios()->avg('nota')) }}
```

## 4.3CREAR FORMATO INTERVAL

```
public function up(): void
    {

        Grammar::macro('typeInterval', function () {
            return 'interval';
        });

se añade eso  justo antes del Schema….

El formato debe ser de la siguiente forma:
```

```
$table->addColumn('interval', 'duracion');

El grammar que hay que añadir es el siguiente:
use Illuminate\Database\Schema\Grammars\Grammar;
```

Insertar en formato PMT P3M20S = 3 minutos y 20 segundos
P4Y2M3DT3H2M5S = 4 años, 2 meses, 3 dias, 3 horas , 2 minutos y 5 segundos

## 4.4PODER ORDENAR POR FILAS

Paso 1 el index debe quedar tal que asi :

```
public function index(Request $request)
   {
      $order = $request->query('order', 'denominacion');
      $order_dir = $request->query('order_dir', 'asc');
      $articulos = Articulo::with(['iva', 'categoria'])
         ->selectRaw('articulos.*')
         ->leftJoin('categorias', 'articulos.categoria_id', '=', 'categorias.id')
         ->leftJoin('ivas', 'articulos.iva_id', '=', 'ivas.id')
         ->orderBy($order, $order_dir)
         ->paginate(3);
      return view('articulos.index', [
         'articulos' => $articulos,
         'order' => $order,
         'order_dir' => $order_dir,
      ]);
   }
```

con la consulta personalizada.

Crear un archivo helpers.php en app y pegar las funciones :

```
<?php

function order_dir_arrow($order, $order_dir)
{
   return $order == false ? '' : ($order_dir == 'desc' ? '↑' : '↓');
}

function order_dir($order, $order_dir)
{
   return $order == false ? 'asc' : ($order_dir == 'asc' ? 'desc' :
'asc');
}
```

Para que laravel cargue el helper agregar en el composer.json :

```
  "autoload": {
      "psr-4": {
         "App\\": "app/",
         "Database\\Factories\\": "database/factories/",
         "Database\\Seeders\\": "database/seeders/"
      },
      "files": ["app/helpers.php"]
   },
```

sustituimos ese por el que ya tenemos

despues de modificar ponemos en el terminal : composer dump-autoload

## 4.5CREAR ALGO DIRECTAMENTE AL USUARIO REGISTRADO SIN PEDIRLO EN EL FORMULARIO:

```
public function store(Request $request)
       {

           $reserva = new Reserva();
           $reserva->user_id = auth()->id();
           $reserva->vuelo_id = $request->input('vuelo_id');
           $reserva->asiento = $request->input('asiento');
           $reserva->save();

           session()->flash('success', 'cancion creado
correctamente.');
           return redirect()->route('reservas.index');
       }
```

En el store indicamos que el user_id = auth()->id();

## 5.ESTRUCTURAS

### 5.1if ( $loquesea ) { accion } else { accion 2 }

```
if ($cancion->albumes->isEmpty() and $cancion->artistas->isEmpty()) {
           $cancion->delete();
       } else {
           session()->flash('error', 'La categoría tiene artículos.');
       }
```

5.2@FOREACH ($ejemplos as $ejemplo)
@ENDFOREACH

```
<tbody>
                @foreach ($vuelos as $vuelo)
                <tr class="bg-white border-b dark:bg-gray-800
dark:border-gray-700">

                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->codigo}}
                    </th>
                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->aeropuerto_origen->nombre }}
                    </th>
                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->aeropuerto_destino->nombre}}
                    </th>
                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->precio}}
                    </th>
                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->plazas}}
                    </th>
                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->plazasDisponibles()}}
                    </th>
                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->salida}}
                    </th>
                    <th scope="row" class="px-6 py-4 font-medium
text-gray-900 whitespace-nowrap dark:text-white">
                        {{ $vuelo->llegada}}
                    </th>
                    </tr>
                @endforeach
            </tbody>
```

## 6. Errores

SQLSTATE[23503]: Foreign key violation: 7 ERROR: update o delete en «temas» viola la llave foránea «artistas_temas_tema_id_foreign» en la tabla «artistas_temas» DETAIL: La llave (id)=(3) todavía es referida desde la tabla «artistas_temas». DELETE FROM "temas" WHERE "id" = 3

Queremos borrar algo que es clave primaria de una tabla pivote. Para ello debemos borrar primero la tabla pivote.

```
public function destroy(Tema $tema)
   {

  // Eliminar las referencias en la tabla pivote
  DB::table('artistas_temas')->where('tema_id', $tema->id)->delete();

  // Luego eliminar el tema
  $tema->delete();

  // Crear un mensaje de éxito y redirigir
  session()->flash('success', 'Tema eliminado correctamente.');
  return redirect()->route('temas.index');
   }
```

## 7. Funciones

```
 public function nombres_alumnos()
   {
      $registros = Nota::where('asignatura_id', $this->id)->get();
      $nombres_alumnos = [];
      foreach ($registros as $registro) {
         $cod = $registro->alumno_id;
         $n = Alumno::find($cod);

         if (!in_array($n->nombre, $nombres_alumnos)) {
               $nombres_alumnos[] = $n->nombre;
            }

      }
      // Construir la lista de nombres de álbumes
      $lista_nombres_alumnos = '';
      foreach ($nombres_alumnos as $r) {
         $lista_nombres_alumnos .= '<li>' . $r . '</li>';
```

```
        }
        return $lista_nombres_alumnos ? '<ul>' . $lista_nombres_alumnos . '</ul>' : 'Sin
alumnos';
    }


    public function notas_primer_trimestre()
    {
        $canciones = Nota::where('asignatura_id', $this->id)
                        ->where('trimestre', '1')
                        ->get();

        $nombres_canciones = '';
        foreach ($canciones as $cancion) {
            $nombre = $cancion->nota;
            $nombres_canciones .= '<li>' . $nombre . '</li>';
        }
        return $nombres_canciones ? '<ul>' . $nombres_canciones . '</ul>' : 'Sin calificar';
    }

    public function notas_segun_trimestre()
    {
        $canciones = Nota::where('asignatura_id', $this->id)
                        ->where('trimestre', '1')
                        ->get();

        $nombres_canciones = '';
        foreach ($canciones as $cancion) {
            $nombre = $cancion->nota;
            $nombres_canciones .= '<li>' . $nombre . '</li>';
        }
        return $nombres_canciones ? '<ul>' . $nombres_canciones . '</ul>' : 'Sin calificar';
    }

  public function nombres_canciones()
    {
        $canciones = AlbumTema::where('album_id', $this->id)->get();
        $nombres_canciones = '';
        foreach ($canciones as $cancion) {
            $nombre = $cancion->tema_id;

            $al = Tema::find($nombre);
            $nombres_canciones .= '<li>' . $al->titulo . '</li>';
        }
        return $nombres_canciones ? '<ul>' . $nombres_canciones . '</ul>' : 'Sin canciones';
    }

    public function duracion_album()
```

```
    {
        $registros = AlbumTema::where('album_id', $this->id)->get();
        $duracion = 0;

        foreach ($registros as $registro) {
            $cancion = Tema::find($registro->tema_id);
            $tiempo = $cancion->duracion;

            $total_cancion = Carbon::createFromFormat('H:i:s', $tiempo);
            $duracion += $total_cancion->hour * 3600 + $total_cancion->minute * 60 +
$total_cancion->second;
        }

        // Convertir la duración total de segundos a formato minutos:segundos
        $minutos = floor($duracion / 60);
        $segundos = $duracion % 60;

        return sprintf('%02d:%02d', $minutos, $segundos);
    }
```

## 8. Gates Y Politicas

### 1. Definir Gates

**Las gates se definen en el `AuthServiceProvider`. Aquí es donde se registra toda la lógica de autorización.**

**Ejemplo de definición de una gate:**

```
use Illuminate\Support\Facades\Gate;

class AuthServiceProvider extends ServiceProvider
{
    public function boot()
    {
        $this->registerPolicies();

        Gate::define('update-post', function ($user,
$post) {
            return $user->id === $post->user_id;
```

```
        });
    }
}
```

**En este ejemplo, se define una gate llamada `update-post` que verifica si el ID del usuario coincide con el ID del usuario que creó el post.**

**2. Utilizar Gates en los Controladores o Vistas**

**Una vez definida una gate, puedes utilizarla en tus controladores, vistas o cualquier otro lugar de tu aplicación donde necesites verificar la autorización.**

**Ejemplo en un controlador:**

```
public function update(Request $request, Post $post)
{
    if (Gate::allows('update-post', $post)) {
        // El usuario está autorizado a actualizar el
post
        // Lógica para actualizar el post
    } else {
        // El usuario no está autorizado
        abort(403);
    }
}
```

**En este ejemplo, se utiliza `Gate::allows` para verificar si el usuario está autorizado para actualizar el post.**

**3. Métodos de Verificación**

**Laravel proporciona varios métodos para trabajar con gates:**

- `Gate::allows('gate-name', $arguments)`: Retorna `true` si el usuario está autorizado.
- `Gate::denies('gate-name', $arguments)`: Retorna `true` si el usuario no está autorizado.
- `Gate::check('gate-name', $arguments)`: Similar a `allows`, pero más semántico.
- `Gate::forUser($user)->allows('gate-name', $arguments)`: Verifica autorización para un usuario específico.

**4. Definir Políticas**

**Para manejar la lógica de autorización de manera más estructurada, Laravel proporciona las políticas, que son clases dedicadas a definir la autorización para un modelo específico.**

**Generar una política:**

```
php artisan make:policy PostPolicy
```

**Definir la política:**

```
class PostPolicy
{
    public function update(User $user, Post $post)
    {
        return $user->id === $post->user_id;
    }
}
```

**Registrar la política:**

**En el `AuthServiceProvider`:**

```
protected $policies = [
    'App\Models\Post' => 'App\Policies\PostPolicy',
];

public function boot()
{
    $this->registerPolicies();
}
```

#### Usar la política:

```
if ($user->can('update', $post)) {
    // El usuario está autorizado
}
```

#### 1. Verificación de Roles

**Puedes utilizar gates para verificar si un usuario tiene un rol específico.**

```
Gate::define('admin-access', function ($user) {
    return $user->role === 'admin';
});
```

#### Uso en un Controlador

```
public function adminDashboard()
{
    if (Gate::denies('admin-access')) {
        abort(403);
    }
```

```
    // Mostrar el panel de administración
}
```

#### 2. Permiso para Crear Posts

**Verificar si un usuario tiene permiso para crear posts.**

```
Gate::define('create-post', function ($user) {
    return $user->hasPermission('create-post');
});
```

#### Uso en un Controlador

```
public function create()
{
    if (Gate::denies('create-post')) {
        abort(403);
    }

    // Mostrar formulario de creación de post
}
```

### Usando Gates en Vistas Blade

**También puedes utilizar gates directamente en tus vistas Blade para mostrar u ocultar contenido basado en permisos.**

```
@can('update-post', $post)
    <a href="{{ route('posts.edit', $post) }}">Editar
Post</a>
@endcan

@cannot('update-post', $post)
    <p>No tienes permiso para editar este post.</p>
```

```
@endcannot
```

**Resumen**

**Las gates en Laravel son una forma poderosa y flexible de controlar la autorización en tu aplicación. Puedes definirlas en el `AuthServiceProvider` y utilizarlas en cualquier parte de tu aplicación para asegurarte de que los usuarios solo realicen acciones para las que están autorizados. Además, las políticas proporcionan una forma estructurada de manejar la autorización para modelos específicos, lo que facilita la gestión y el mantenimiento de las reglas de autorización en tu aplicación.**

```
('update-post', $post)
    <p>No tienes permiso para editar este post.</p>
@endcannot
```

**Resumen**

**Las gates en Laravel son una forma poderosa y flexible de controlar la autorización en tu aplicación. Puedes definirlas en el `AuthServiceProvider` y utilizarlas en cualquier parte de tu aplicación para asegurarte de que los usuarios solo realicen acciones para las que están autorizados. Además, las políticas proporcionan una forma estructurada de manejar la autorización para modelos específicos, lo que facilita la gestión y el mantenimiento de las reglas de autorización en tu aplicación.**