Projecte GLOBAL: VideosApp

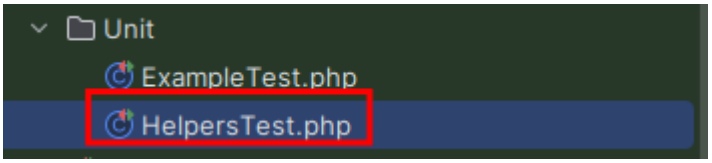
2n SPRINT

Professor: Jordi Vega
Assignatures: M7, M8 i M9
Nom: Manuel Steven
Cognoms: Romero Vargas

Índex

Corregir els errors del 1r sprint.

El primer error que tenia era que el test **HelpersTest** no es trobaba en el directori **Unit**, en el meu cas el tenia en el directori **Features**, aixi que com a primer pas l'he mogut al directori **Unit.**



Un altre petit error que podiem trobar en el nostre test, es que no testejaba suficients camps, aixi alguns més com el current_team_id, la contrasenya, el user_id, el personal team.

```php
public function test_can_create_default_user()
{
    // Llamamos al helper para crear el usuario por defecto
    $user = UserHelper::createDefaultUser();

    // Verificamos que el usuario se haya creado en la base de datos
    $this->assertDatabaseHas( table: 'users', [
        'name' => config( key: 'users.default_user.name'),
        'email' => config( key: 'users.default_user.email'),
        'current_team_id' => $user->current_team_id, // Usamos el equipo asignado dinámicamente
    ]);

    // Verificamos que la contraseña almacenada sea válida
    $this->assertTrue(Hash::check(
        config( key: 'users.default_user.password'),
        $user->password
    ));

    // Verificamos que el equipo personal se haya creado correctamente
    $this->assertDatabaseHas( table: 'teams', [
        'name' => $user->name . "'s Team",
        'user_id' => $user->id,
        'personal_team' => true,
    ]);
}
```

Seguirem el mateix procediment en el test del usuari professor.

```php
Steven *
public function test_can_create_default_teacher()
{
    // Llamamos al helper para crear el profesor por defecto
    $teacher = UserHelper::createDefaultTeacher();

    // Verificamos que el profesor se haya creado en la base de datos
    $this->assertDatabaseHas( table: 'users', [
        'name' => config( key: 'users.default_teacher.name'),
        'email' => config( key: 'users.default_teacher.email'),
        'current_team_id' => $teacher->current_team_id, // Usamos el equipo asignado dinámicamente
    ]);

    // Verificamos que la contraseña almacenada sea válida
    $this->assertTrue(Hash::check(
        config( key: 'users.default_teacher.password'),
        $teacher->password
    ));

    // Verificamos que el equipo personal se haya creado correctamente
    $this->assertDatabaseHas( table: 'teams', [
        'name' => $teacher->name . "'s Team",
        'user_id' => $teacher->id,
        'personal_team' => true,
    ]);
}
```

Afegirem el current_team_id al **config/users.php** per a que ens agafi la verificació correctament.

```php
php users.php
<?php

return [
    'default_user' => [
        'name' => env( key: 'DEFAULT_USER_NAME', default: 'Robin'),
        'email' => env( key: 'DEFAULT_USER_EMAIL', default: 'robin@iesebre.com'),
        'password' => env( key: 'DEFAULT_USER_PASSWORD', default: 'RobinHood'),
        'current_team_id' => env( key: 'DEFAULT_USER_CURRENT_TEAM_ID', default: 1),
    ],
    'default_teacher' => [
        'name' => env( key: 'DEFAULT_TEACHER_NAME', default: 'Batman'),
        'email' => env( key: 'DEFAULT_TEACHER_EMAIL', default: 'batman@iesebre.com'),
        'password' => env( key: 'DEFAULT_TEACHER_PASSWORD', default: 'TheDarkKnight'),
        'current_team_id' => env( key: 'DEFAULT_TEACHER_CURRENT_TEAM_ID', default: 2),
    ],
];
```

Descomentari linies de db_connection i db_database

Ens haurem de dirigir al fitxer **phpunit.xml** i descomentar les següents linies per tal de que els testos utilitzi una base de dades temporal i no afecti a la base de dades real.

```
<env name="DB_CONNECTION" value="sqlite"/>
<env name="DB_DATABASE" value=":memory:"/>
```

Crear la migració de videos

Utilitzarem l'ordre **php artisan make:migration** per a crear una migració per a la nostra taula de videos.

```
alumnat@steven:~/Documents/VideosAppSteven$ php artisan make:migration create_videos_table --create=videos

  INFO  Migration [database/migrations/2025_01_16_145657_create_videos_table.php] created successfully.
```

Un cop creada la migració haurem d'establir els diferents camps demanats en l'enunciat en el format corresponent. (series_id per ara no es una clau foranea ja que la taula de series la crearem en el següent sprint)

```
public function up()
{
    Schema::create( table: 'videos', function (Blueprint $table) {
        $table->id(); // id autoincremental
        $table->string( column: 'title'); // titulo del video
        $table->text( column: 'description'); // descripción, opcional
        $table->string( column: 'url'); // URL del video
        $table->timestamp( column: 'published_at')->nullable(); // fecha de publicación, opcional
        $table->string( column: 'previous')->nullable(); // referencia al video anterior
        $table->string( column: 'next')->nullable(); // referencia al video siguiente
        $table->unsignedBigInteger( column: 'series_id'); // referencia a la serie, sin clave foránea
        $table->timestamps(); // created_at y updated_at

        // Clave foránea será agregada más adelante
        // $table->foreign('series_id')->references('id')->on('series')->cascadeOnDelete();
    });
```

Crear controlador de Videos

Per a crear el controlador **VideosController** utilitzarem l'odre **php artisan make:controller**.

Controlador de videos, VideosController amb les funcions testedBy i el show.

```
alumnat@steven:~/Documents/VideosAppSteven$ php artisan make:controller VideosController

  INFO  Controller [app/Http/Controllers/VideosController.php] created successfully.
```

Dins del controlador crearem la funció de show que ens trobara el video segons la seva id, y en cas de que la trobe la mostrarà en l'arxiu **show.blade.php** que es trobara en el directori **videos** dins de **resources/views**, si el video no existeix retorna un error 404.

```php
public function show($id)
{
    // Busca el vídeo per ID
    $video = Video::find($id);

    // Si el vídeo no existeix, retorna un error 404
    if (!$video) {
        abort( code: 404,  message: 'Video not found');
    }

    // Retorna la vista del vídeo amb la informació
    return view( view: 'videos.show', compact( var_name: 'video'));
}
```

En el tested by verifico si el video existeix també, sino existeix retorno a l'usuari a la pàgina 404.

```php
public function testedBy($id)
{
    // Busca el video por ID
    $video = Video::find($id);

    // Si el video no existe, devuelve un error 404
    if (!$video) {
        return response()->json(['error' => 'Video not found'],  status: 404);
    }
}
```

Crear model de Videos

Crearem el model de videos en l'ordre **php artisan make:model**

```
alumnat@steven:~/Documents/VideosAppSteven$ php artisan make:model Video

  INFO  Model [app/Models/Video.php] created successfully.
```

En el model primer que tot afegirem els camps que podes ser assignats.

```php
class Video extends Model
{
    use HasFactory;

    // Campos que pueden ser asignados de forma masiva
    no usages
    protected $fillable = [
        'title',          // Título del video
        'description',    // Descripción del video
        'url',            // URL del video
        'previous',
        'next',
        'series_id', // Asegúrate de incluir este campo
        'published_at',   // Fecha de publicación
    ];
```

En el camp de published_at crearem una funció **getFormattedPublishedAtAttribute()** que ens retornara la data formatejada, de manera que el resultat serà. per exemple: “13 de gener de 2025”

```php
no usages
protected $dates = ['published_at'];

/**
 * Devuelve la fecha publicada formateada (por ejemplo, "13 de enero de 2025").
 *
 * @return string|null
 */
no usages  Steven
public function getFormattedPublishedAtAttribute(): ?string
{
    if ($this->published_at) {
        return Carbon::parse($this->published_at)->isoFormat( format: 'D [de] MMMM [de] YYYY');
    }
    return null;
}
```

Crearem també una funció anomenada **getFormattedForHumansPublishedAtAttribute()**, que ens retornara la data publicada en un format llegible, per exemple (fa 2 hores)

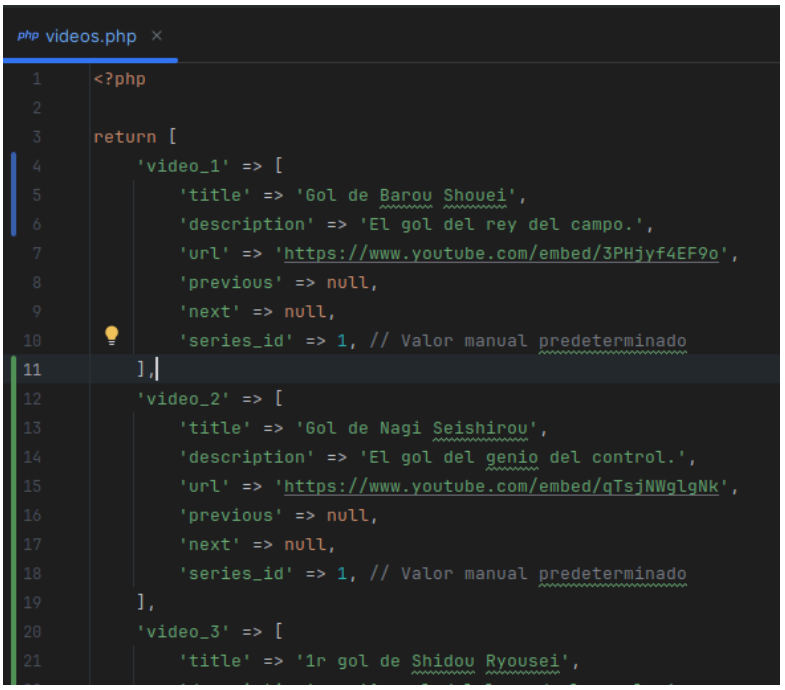
```php
/**
 * Devuelve la fecha publicada en un formato legible (por ejemplo, "hace 2 horas").
 *
 * @return string|null
 */
no usages  Steven
public function getFormattedForHumansPublishedAtAttribute(): ?string
{
    if ($this->published_at) {
        return Carbon::parse($this->published_at)->diffForHumans();
    }
    return null;
}
```

Per finalitzar crearem una funció anomenada **getPublishedAtTimestampAttribute()** que ens retorna el Unix timestamp de la data publicada.

```php
/**
 * Devuelve el Unix timestamp de la fecha publicada.
 *
 * @return int|null
 */
no usages  Steven
public function getPublishedAtTimestampAttribute(): ?int
{
    if ($this->published_at) {
        return Carbon::parse($this->published_at)->timestamp;
    }
    return null;
}
```

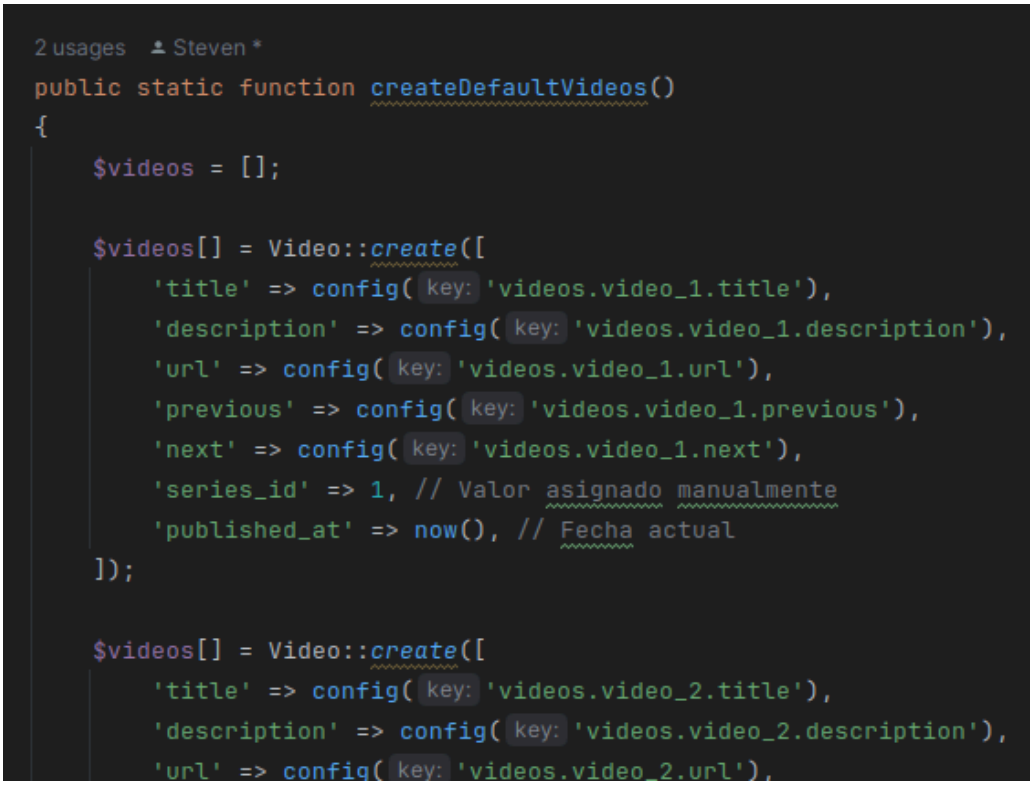
Crear helper de videos

En el directori **config** crearem un arxiu videos.php on tindrem la informació dels diferents videos que utilitzarem més endavant.



```php
<?php

return [
    'video_1' => [
        'title' => 'Gol de Barou Shouei',
        'description' => 'El gol del rey del campo.',
        'url' => 'https://www.youtube.com/embed/3PHjyf4EF9o',
        'previous' => null,
        'next' => null,
        'series_id' => 1, // Valor manual predeterminado
    ],
    'video_2' => [
        'title' => 'Gol de Nagi Seishirou',
        'description' => 'El gol del genio del control.',
        'url' => 'https://www.youtube.com/embed/qTsjNWglgNk',
        'previous' => null,
        'next' => null,
        'series_id' => 1, // Valor manual predeterminado
    ],
    'video_3' => [
        'title' => '1r gol de Shidou Ryousei',
```

Crearem el VideoHelper y dins una funció per a crear els diferents videos, aixo ens ajudarà en el procés dels testos on verificarem que els videos es puguin crear correctament i en el seeder on tindrem els videos per defecte a crear en la nostra aplicació.



```php
2 usages  Steven *
public static function createDefaultVideos()
{
    $videos = [];

    $videos[] = Video::create([
        'title' => config( key: 'videos.video_1.title'),
        'description' => config( key: 'videos.video_1.description'),
        'url' => config( key: 'videos.video_1.url'),
        'previous' => config( key: 'videos.video_1.previous'),
        'next' => config( key: 'videos.video_1.next'),
        'series_id' => 1, // Valor asignado manualmente
        'published_at' => now(), // Fecha actual
    ]);

    $videos[] = Video::create([
        'title' => config( key: 'videos.video_2.title'),
        'description' => config( key: 'videos.video_2.description'),
        'url' => config( key: 'videos.video_2.url'),
```

Afegir els seeders

En el projecte anterior en el **DatabaseSeeder.php** ja havíem fer la crida al helper que crea els usuaris per defecte, ara farem la crida a el helper de vídeos per a crear els vídeos predeterminats.

DatabaseSeeder.php

```php
<?php

namespace Database\Seeders;

use App\Helpers\UserHelper;
use App\Helpers\VideoHelper;
use App\Models\User;
// use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;

no usages  Steven *
class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     */
    Steven *
    public function run(): void
    {
        // User::factory(10)->withPersonalTeam()->create();

        // Crear el usuario por defecto
        UserHelper::createDefaultUser();

        // Crear el profesor por defecto
        UserHelper::createDefaultTeacher();

        // Crear el video por defecto
        VideoHelper::createDefaultVideo();
    }
}
```

Crear Layout de videos

Crearem un component anomenat **VideosAppLayout** en l'ordre **php artisan make:component.**

```
alumnat@steven:~/Documents/VideosAppSteven$ php artisan make:component VideosAppLayout

  INFO  Component [app/View/Components/VideosAppLayout.php] created successfully.

  INFO  View [resources/views/components/videos-app-layout.blade.php] created successfully.
```

Dins de **app/View/components** tindrem el layout que ens dirigira a la vista de videos-app-layout que es troba dins del directori **components**

VideosAppLayout.php

```php
class VideosAppLayout extends Component
{
    /**
     * Create a new component instance.
     */
    no usages
    public function __construct()
    {
        //
    }

    /**
     * Get the view / contents that represent the component.
     */
    no usages
    public function render(): View|Closure|string
    {
        return view( view: 'components.videos-app-layout');
    }
}
```

En el layout tindrem l'estructura bàsica de la nostra pàgina de videos, y fent us d'un slot, mostrarem en la vista show el contingut de cada video.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ $title ?? 'Videos App' }}</title>
    @vite('resources/css/app.css') {{-- Tailwind CSS --}}
</head>
<body class="flex flex-col min-h-screen bg-gray-300 text-gray-800 font-sans">

<!-- Main Content -->
<main class="flex-grow container mx-auto px-4 py-6">
    {{ $slot }}
</main>
</body>
</html>
```

Crear la ruta del show

En **routes/web.php** haurem d'afegir la ruta al show de les vistes de videos, de manera que podrem anar visualitzant cada video segons la seva id.

web.php

```php
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\VideosController;


Route::get( uri: '/', function () {
    return view( view: 'welcome');
});

Route::get( uri: 'videos/{id}', [VideosController::class, 'show'])->name( name: 'videos.show');

```

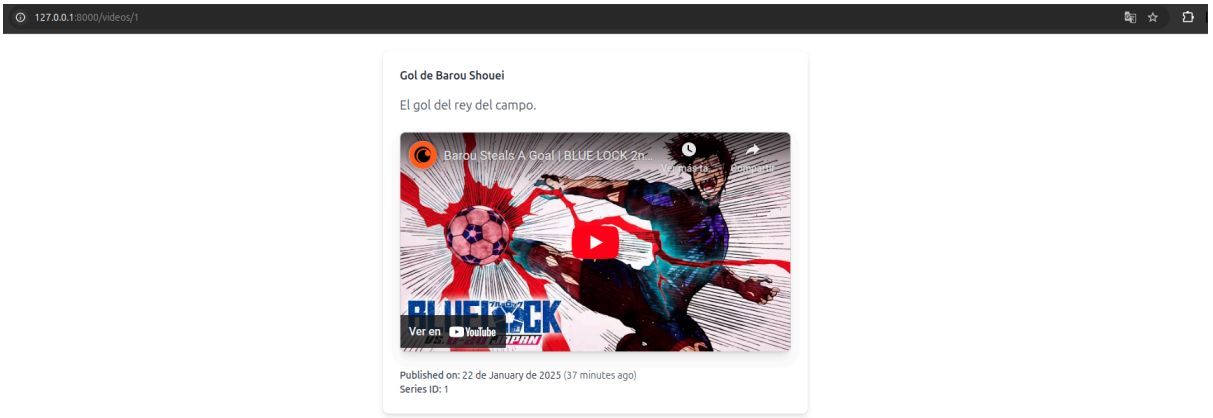
Crearem la vista del show

Crearem la vista del show de videos, de manera que podrem veure la informació del nostre video (títol, descripció, data de publicació formatada, series id) i el mateix video mitjançant un iframe. Per finalitzar afegirem uns estils bàsics en tailwindcss, tot i que no tindrà res especial ja que en els següents sprints ja ens centrarem en pulir els estils.

show.blade.php

```blade
<x-videos-app-layout title="Video: {{ $video->title }}">
    <div class="bg-white shadow-md rounded-lg p-6">
        <h1 class="text-3xl font-semibold text-gray-800 mb-4">{{ $video->title }}</h1>
        <p class="text-gray-600 text-lg mb-6">{{ $video->description }}</p>

        <div class="flex justify-center mb-6">
            <iframe
                src="{{ $video->url }}"
                class="rounded-lg shadow-lg border border-gray-300"
                width="560"
                height="315"
                frameborder="0"
                allow="autoplay; encrypted-media"
                allowfullscreen>
            </iframe>
        </div>

        <div class="text-sm text-gray-500">
            <p>
                <span class="font-medium text-gray-700">Published on:</span>
                <span class="text-gray-800">{{ $video->formatted_published_at }}</span>
                (<span class="text-gray-600">{{ $video->formatted_for_humans_published_at }}</span>)
            </p>
            <p>
                <span class="font-medium text-gray-700">Series ID:</span>
                <span class="text-gray-800">{{ $video->series_id }}</span>
            </p>
        </div>
    </div>
</x-videos-app-layout>

```

Podrem veure com al crear un video, podem accedir correctament segons la seva id, i podem veure com en el iframe es mostra correctament.



Afegir test de creació de videos

Dins del nostre HelpersTest farem la crida a el helper de videos que hem creat anteriorment.

HelpersTest.php

```php
<?php

namespace Tests\Unit;

use App\Helpers\UserHelper;
use App\Helpers\VideoHelper;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Support\Facades\Hash;
use Tests\TestCase;
```

I crearem un petit test que creará un video y verificarà que es creen els diferents camps, com el títol, descripció, url, series id i la data de publicació.

```php
public function test_can_create_default_video()
{
    // Llamamos al helper para crear los videos por defecto
    $videos = VideoHelper::createDefaultVideos();

    // Seleccionamos el primer video creado
    $video = $videos->first();

    // Verificamos que el video se haya creado correctamente en la base de datos
    $this->assertDatabaseHas( table: 'videos', [
        'title' => config( key: 'videos.video_1.title'),
        'description' => config( key: 'videos.video_1.description'),
        'url' => config( key: 'videos.video_1.url'),
        'series_id' => 1, // Valor esperado manualmente
    ]);

    // Verificamos el formato personalizado de la fecha publicada
    $formattedDate = $video->formatted_published_at;
    $this->assertNotNull($formattedDate);
    $this->assertEquals(now()->isoFormat( format: 'D [de] MMMM [de] YYYY'), $formattedDate);
}
```

Crear VideosTest al directori tests/Unit

Crearem un test dins de VideosTest que comprovi que la formatació dels videos és correcta, l'anomenarem **can_get_formatted_published_at_date()**.

```
Steven *
public function test_can_get_formatted_published_at_date()
{

    // Crear un vídeo amb una data de publicació
    $video = Video::factory()->create([
        'published_at' => Carbon::parse( time: '2025-01-17 14:00:00'),
    ]);

    // Comprovar que el format és correcte
    $this->assertEquals( expected: '17 de enero de 2025', $video->formatted_published_at);
}
```

Al executar el test en primera instancia podrem veure que ens retorna un error, i es que al haver ficat la data en espanyol, realment no coincideix en el format definit pel test.

```
alumnat@steven:~/Documents/VideosAppSteven$ php artisan test --filter=VideosTest

  FAIL  Tests\Unit\VideosTest
  × can get formatted published at date

  FAILED  Tests\Unit\VideosTest > can get formatted published at date
  Failed asserting that two strings are equal.
  -'17 de enero de 2025'
  +'17 de January de 2025'


  at tests/Unit/VideosTest.php:24
     20 |             'published_at' => Carbon::parse('2025-01-17 14:00:00'),
     21 |         ]);
     22 |
     23 |         // Comprovar que el format és correcte
  ➜  24 |         $this->assertEquals('17 de enero de 2025', $video->formatted_published_at);
     25 |     }
     26 |
     27 | //    public function test_can_get_formatted_published_at_date()
     28 | //    {

  1   tests/Unit/VideosTest.php:24


  Tests:    1 failed (1 assertions)
  Duration: 0.16s
```

Podriem simplement cambiar el test a angles, o també tenim l'opció de establir l'idioma de Carbon a espanyol.

```
Steven
public function test_can_get_formatted_published_at_date()
{
    // Configurar idioma a espanyol
    Carbon::setLocale( locale: 'es');

    // Crear un vídeo amb una data de publicació
    $video = Video::factory()->create([
        'published_at' => Carbon::parse( time: '2025-01-17 14:00:00'),
    ]);

    // Comprovar que el format és correcte
    $this->assertEquals( expected: '17 de enero de 2025', $video->formatted_published_at);
}
```

Podrem veure com realment el test es realitza sense problemes.

```
alumnat@steven:~/Documents/VideosAppSteven$ php artisan test --filter=VideosTest

  PASS  Tests\Unit\VideosTest
  ✓ can get formatted published at date

  Tests:    1 passed (1 assertions)
  Duration: 0.16s
```

Crearem un altre test que comprovara que es pot crear un video sense data de publicació, anomenarem al test

can_get_formatted_published_at_date_when_not_published().

```php
public function test_can_get_formatted_published_at_date_when_not_published()
{
    // Crear un vídeo sense data de publicació
    $video = Video::factory()->create([
        'published_at' => null,
    ]);

    // Comprovar que la propietat retorna null
    $this->assertNull($video->formatted_published_at);
}
```

Realment com aquest test no té cap misteri podem executar-lo i veure com no hi ha cap tipus de problema.

```
alumnat@steven:~/Documents/VideosAppSteven$ php artisan test --filter=VideosTest

  PASS  Tests\Unit\VideosTest
  ✓ can get formatted published at date when not published

  Tests:    1 passed (1 assertions)
  Duration: 0.16s
```

Crear VideosTest al directori tests/Feature

Crearem un test anomenat **users_can_view_videos()** que verifica que un cop creat el video, les persones podran accedir a la pàgina del show, és a dir, que podràn visualitzar els videos creats correctament.

```php
public function test_users_can_view_videos()
{
    $video = Video::factory()->create();

    $response = $this->get(route( name: 'videos.show', $video->id));

    $response->assertStatus( status: 200);
    $response->assertViewIs( value: 'videos.show');
    $response->assertViewHas('video', $video);
}
```

I podem veure com realment s'executa el test sense cap tipus de problema, perque com hem verificat anteriorment, realment podem accedir a la pàgina de show.

```
Calumnat@steven:~/Documents/VideosAppSteven$ php artisan test --filter=VideosTest

  PASS  Tests\Feature\VideosTest
  ✓ users can view videos

  Tests:    1 passed (3 assertions)
  Duration: 0.17s
```

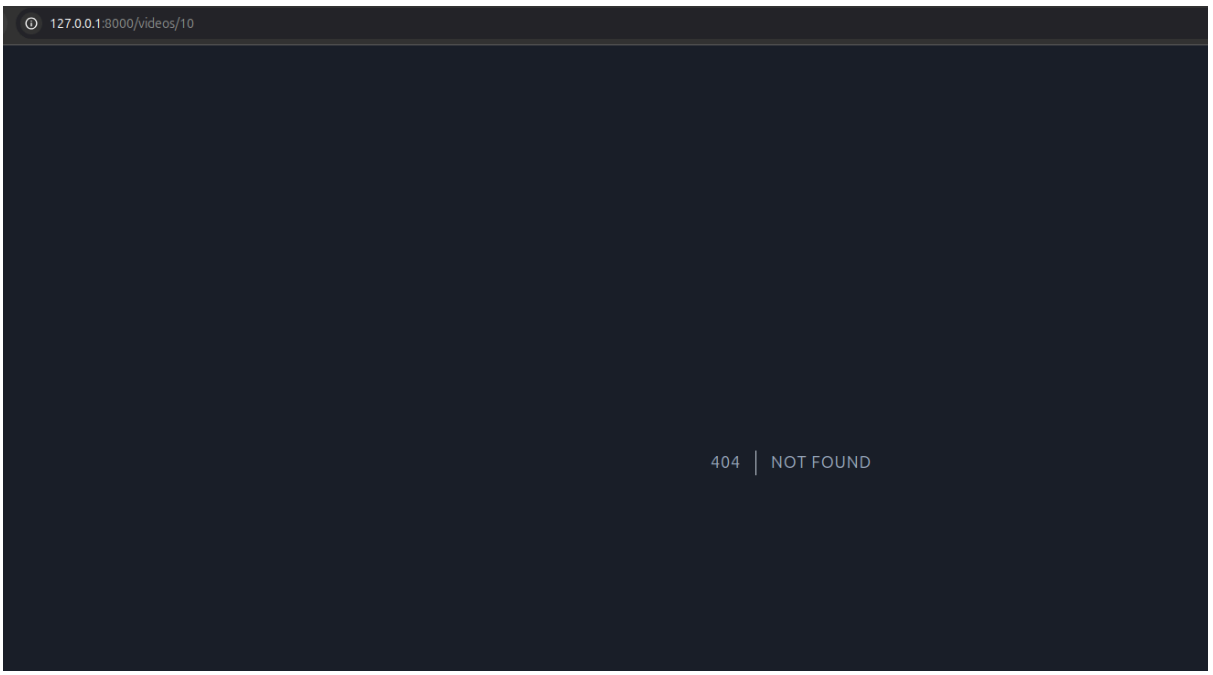
També en un test que anomenarem **users_cannot_view_not_existing_videos()** que verificarà que si no hi han videos o, en cas de que hagues intentes accedir a un video inexistent, ens apareixera l'error 404

```
Steven *
public function test_users_cannot_view_not_existing_videos()
{
    $response = $this->get(route( name: 'videos.show',  parameters: 9999));

    $response->assertStatus( status: 404);
}
```

I podem veure com realment ens

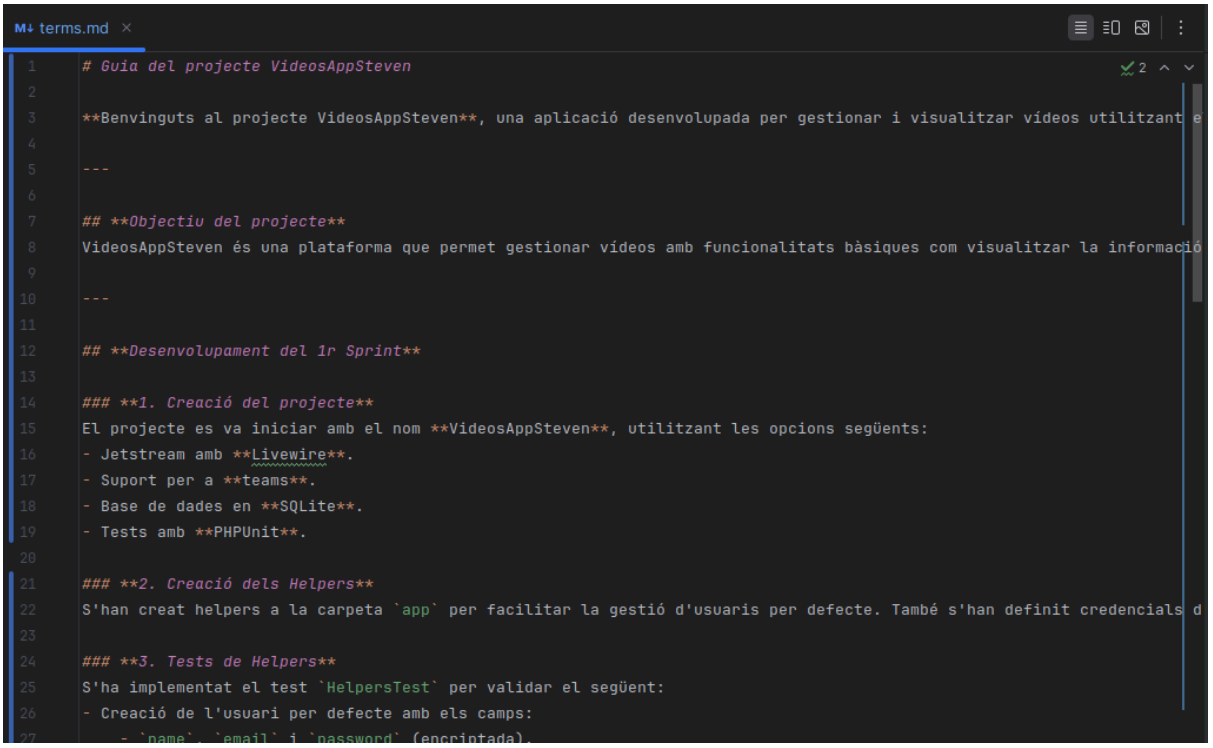
```
alumnat@steven:~/Documents/VideosAppSteven$ php artisan test --filter=VideosTest

  PASS  Tests\Feature\VideosTest
  ✓ users cannot view not existing videos

  Tests:    1 passed (1 assertions)
  Duration: 0.16s
```

Podrem veure com realment al accedir a una pàgina ens retorna el error **404 | NOT FOUND**



Afegir guia del projecte

Afegirem a **resources/markdown/terms** una petita guia sobre que tracta el projecte i que hem fet als dos sprints.



Instal·lació i execució Larastan

Per a instal·lar larastan en el nostre projecte executarem l'ordre **composer require --dev "larastan/larastan3.0"**

```
alumnat@steven:~/Documents/VideosAppSteven$ composer require --dev "larastan/larastan:^3.0"
./composer.json has been updated
Running composer update larastan/larastan
Loading composer repositories with package information
Updating dependencies
Nothing to modify in lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

  INFO  Discovering packages.

  laravel/fortify ..... DONE
  laravel/jetstream ..... DONE
  laravel/pail ..... DONE
  laravel/sail ..... DONE
  laravel/sanctum ..... DONE
  laravel/tinker ..... DONE
  livewire/livewire ..... DONE
  nesbot/carbon ..... DONE
  nunomaduro/collision ..... DONE
  nunomaduro/termwind ..... DONE

83 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

  INFO  No publishable resources for tag [laravel-assets].

Found 2 security vulnerability advisories affecting 2 packages.
Run "composer audit" for a full list of advisories.
```

Crearem un arxiu **phpstan.neon** que incluirá les extensions de **larastan** i **carbon**.

```
alumnat@steven:~/Documents/VideosAppSteven$ cat phpstan.neon
includes:
    - vendor/larastan/larastan/extension.neon
    - vendor/nesbot/carbon/extension.neon

parameters:
    paths:
        - app/
        - routes/
        - tests/
    # Level 10 is the highest level
    level: 5
```

Per a executar un analisis, executarem l'ordre **./vendor/bin/phpstan analyse**

```
alumnat@steven:~/Documents/VideosAppSteven$ ./vendor/bin/phpstan analyse
Note: Using configuration file /home/alumnat/Documents/VideosAppSteven/phpstan.neon.
 24/24 [============================] 100%
```

Correció dels errors trobats

Error 1

Podrem veure com ens troba un error en el directori **Actions/Jetstream/DeleteUser.php**, en el qual troba una propietat indefinida en el projecte.

```
------ ---------------------------------------------------------------------------------------
 Line   Actions/Jetstream/DeleteUser.php
------ ---------------------------------------------------------------------------------------
 40     Access to an undefined property App\Models\User::$ownedTeams.
        🪪  property.notFound
        💡 Learn more: https://phpstan.org/blog/solving-phpstan-access-to-undefined-property
------ ---------------------------------------------------------------------------------------
```

Haurem de definir-la manualment en un comentari, per a que larastan la trobi.

```
/**
 * @property-read \Illuminate\Database\Eloquent\Collection|\App\Models\Team[] $ownedTeams
 */
```

Error 2

En el segon error el trobem en el **Actions/Jetstream/InviteTeamMember.php**, error en el qual hi ha un parametre que no identifica o no troba correctament.

```
Line   Actions/Jetstream/InviteTeamMember.php
37     Parameter #1 $invitation of class Laravel\Jetstream\Mail\TeamInvitation constructor expects Laravel\Jetstream\TeamInvitation, Illuminate\Database\Eloquent\Model given.
       🪪  argument.type
```

Per a solucionar-ho haurem de afegir un comentari per a que larastan trobe la variable **invitation**

```
© InviteTeamMember.php ×
20    class InviteTeamMember implements InvitesTeamMembers
25        public function invite(User $user, Team $team, string $email, ?string $rol
31            InvitingTeamMember::dispatch($team, $email, $role);
32
33            /** @var TeamInvitationModel $invitation */
34            $invitation = $team->teamInvitations()->create([
35                'email' => $email,
36                'role' => $role,
37            ]);
38
39            Mail::to($email)->send(new TeamInvitation($invitation));
40        }
41
```

Error 3

Ens trobarà un error a l'acció **RemoveTeamMember**, on ens dira que ha trobat una propietat indefinida.

```
------ ------------------------------------------------------------------------------
 Line   Actions/Jetstream/RemoveTeamMember.php
------ ------------------------------------------------------------------------------
 45     Access to an undefined property Illuminate\Database\Eloquent\Model::$id.
        🪪  property.notFound
        💡 Learn more: https://phpstan.org/blog/solving-phpstan-access-to-undefined-property
------ ------------------------------------------------------------------------------
```

Per a solucionar aquest error haurem de definir la propietat **id (i altres necessaries)** tant en el model de **user.php** com en el **team.php**

User.php

```php
<?php

namespace App\Models;

// use Illuminate\Contracts\Auth\MustVerifyEmail;
use ...

/**
 * @property-read \Illuminate\Database\Eloquent\Collection|\App\Models\Team[] $ownedTeams
 */

/**
 * @property int $id
 * @property int|null $current_team_id
 * @property-read \App\Models\Team|null $team
 * @property-read \Illuminate\Database\Eloquent\Collection|\App\Models\Team[] $ownedTeams
 */
```

Team.php

```php
<?php

namespace App\Models;

use ...

/**
 * @property int $id
 * @property int $user_id
 * @property bool $personal_team
 * @property-read \App\Models\User $owner
 * @property-read \Illuminate\Database\Eloquent\Collection|\App\Models\User[] $members
 */
```

Error 4

En el model de Team.php podrem veure com ens troba un error en el tipus de propietat definida en la línia 30.

```
------ ------------------------------------------------------------------------------------------------------------------------------------------------------
Line   Models/Team.php
------ ------------------------------------------------------------------------------------------------------------------------------------------------------
30     PHPDoc type array<int, string> of property App\Models\Team::$fillable is not covariant with PHPDoc type list<string> of overridden
       property Illuminate\Database\Eloquent\Model::$fillable.
       🪪  property.phpDocType
       💡 You can fix 3rd party PHPDoc types with stub files:
       💡 https://phpstan.org/user-guide/stub-files
```

El que haurem de fer per a solucionar aquest error serà, cambiar el comentari **array<int, string>** per **list<string>** per a que larastan o pugui llegir bé i sense detectar problemes.

```php
class Team extends JetstreamTeam
{
    /** @use HasFactory<\Database\Factories\TeamFactory> */
    use HasFactory;

    /**
     * The attributes that are mass assignable.
     *
     * @var list<string>
     */
```

Error 5

En aquest error també podem trobar el mateix cas anterior, de manera que seguirem el mateix procediment de ficar **list<string>** per a solucionar el error.

```php
class User extends Authenticatable
{
    use HasApiTokens;

    /** @use HasFactory<\Database\Factories\UserFactory> */
    use HasFactory;
    use HasProfilePhoto;
    use HasTeams;
    use Notifiable;
    use TwoFactorAuthenticatable;

    /**
     * The attributes that are mass assignable.
     *
     * @var list<string>
     */
```

Error 6

En aquest error, literalment podrem trobar el mateix error en la mateixa solució que el anterior punt.

```
------ -----------------------------------------------------------------------------------------------------------------
Line   Models/User.php
------ -----------------------------------------------------------------------------------------------------------------
54     PHPDoc type array<int, string> of property App\Models\User::$hidden is not covariant with PHPDoc type list<string> of overridden property
       Illuminate\Database\Eloquent\Model::$hidden.
       🪪  property.phpDocType
       💡 You can fix 3rd party PHPDoc types with stub files:
       💡 https://phpstan.org/user-guide/stub-files
```

```php
/**
 * The attributes that should be hidden for serialization.
 *
 * @var list<string>
 */
protected $hidden = [
    'password',
    'remember_token',
    'two_factor_recovery_codes',
    'two_factor_secret',
];
```

Error 6

Un altre cop, el mateix error, en la mateixa solucio utilitzant list en comptes d'array.

```
------ -----------------------------------------------------------------------------------------------------------------
Line   Models/User.php
------ -----------------------------------------------------------------------------------------------------------------
66     PHPDoc type array<int, string> of property App\Models\User::$appends is not covariant with PHPDoc type list<string> of overridden property
       Illuminate\Database\Eloquent\Model::$appends.
       🪪  property.phpDocType
       💡 You can fix 3rd party PHPDoc types with stub files:
       💡 https://phpstan.org/user-guide/stub-files
```

```php
/**
 * The accessors to append to the model's array form.
 *
 * @var list<string>
 */
no usages
protected $appends = [
    'profile_photo_url',
];
```

Error 8

En aquest error podrem veure com tenim una propietat que en principi larastan la detecta com a que no existeix.

```
------ ------------------------------------------------------
 Line   Models/User.php
------ ------------------------------------------------------
 67     Property 'profile_photo_url' does not exist in model.
        🪪  rules.modelAppends
------ ------------------------------------------------------
```

Per a solucionar aquest petit problema bastarà en actualitzar el composer.

```
alumnat@steven:~/Documents/VideosAppSteven$ composer update
Loading composer repositories with package information
Updating dependencies
Lock file operations: 4 installs, 44 updates, 0 removals
  - Upgrading egulias/email-validator (4.0.2 => 4.0.3)
  - Upgrading laravel/fortify (v1.24.5 => v1.25.3)
  - Upgrading laravel/framework (v11.33.2 => v11.39.0)
  - Upgrading laravel/jetstream (v5.3.3 => v5.3.4)
  - Upgrading laravel/pint (v1.18.2 => v1.20.0)
  - Upgrading laravel/prompts (v0.3.2 => v0.3.3)
  - Upgrading laravel/sail (v1.38.0 => v1.40.0)
  - Upgrading laravel/sanctum (v4.0.4 => v4.0.7)
  - Upgrading laravel/serializable-closure (v2.0.0 => v2.0.1)
  - Upgrading league/commonmark (2.5.3 => 2.6.1)
  - Locking league/uri (7.5.1)
  - Locking league/uri-interfaces (7.5.0)
  - Upgrading livewire/livewire (v3.5.12 => v3.5.18)
  - Upgrading mobiledetect/mobiledetectlib (4.8.06 => 4.8.09)
```

Error 9

Ens trobarem en un error en el controlador de Videos al metode testedBy()

```
Line   app/Http/Controllers/VideosController.php
42     Method App\Http\Controllers\VideosController::testedBy() should return Illuminate\Http\JsonResponse but return statement is missing.
       return.missing
```

El problema es deu a que el metode **testedBy** està definit per a retornar un objecte de tipus **\Illuminate\Http\JsonResponse**, pero no té una instrucció **return** en el cas en que es trobi un video.

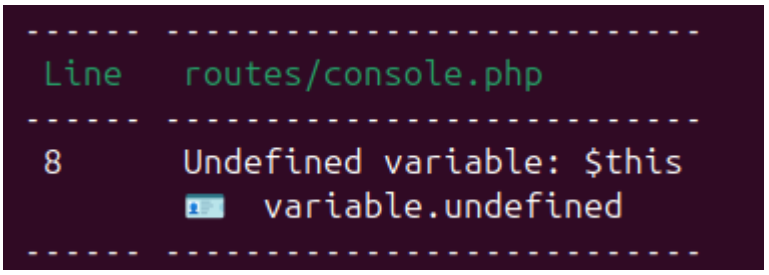
```php
/**
 * Retorna una lista de testers asociados al video.
 *
 * @param int $id
 * @return \Illuminate\Http\JsonResponse
 */
no usages  Steven *
public function testedBy($id)
{
    // Busca el video por ID
    $video = Video::find($id);

    // Si el video no existe, devuelve un error 404
    if (!$video) {
        return response()->json(['error' => 'Video not found'], status: 404);
    }
}
```

Per solucionar aquest problema, ens assegurar-em que el mètode sempre retorni una resposta JSON, fins i tot en el cas que el vídeo existeixi.

```php
public function testedBy($id)
{
    // Busca el video por ID
    $video = Video::find($id);

    // Si el video no existe, devuelve un error 404
    if (!$video) {
        return response()->json(['error' => 'Video not found'], status: 404);
    }

    // Devuelve una lista ficticia de testers asociados al video
    // (sustituir este arreglo por la lógica real que desees implementar)
    $testers = [
        ['id' => 1, 'name' => 'Tester 1', 'email' => 'tester1@example.com'],
        ['id' => 2, 'name' => 'Tester 2', 'email' => 'tester2@example.com'],
    ];

    // Devuelve los testers en formato JSON
    return response()->json(['testers' => $testers], status: 200);
}
```

Error 10

L'error que esteu veient és al **fitxer routes/console.php** està intentant utilitzar la variable **this** fora d'una classe. A Laravel, els fitxers a la carpeta routes no estan dins de classes, per la qual cosa $this no pot ser usat directament.



Per solucionar aquest error, haurem d'usar **Artisan::command** d'una manera que no depengui de **$this** per trucar a mètodes com a comment.

Error 11

En el test de videos podrem observar com no troba la propietat **id** definida correctament.

```
------ ------------------------------------------------------------------------------
 Line   tests/Feature/VideosTest.php
------ ------------------------------------------------------------------------------
 25     Access to an undefined property Illuminate\Database\Eloquent\Model::$id.
        🪪  property.notFound
        💡 Learn more: https://phpstan.org/blog/solving-phpstan-access-to-undefined-property
------ ------------------------------------------------------------------------------
```

Per a solucionar aquest error anirem al model de videos i definirem totes les propietats que haguem creat, per evitar qualsevol tipus d'error.

```php
/**
 * App\Models\Video
 *
 * @property int $id
 * @property string $title
 * @property string $description
 * @property string $url
 * @property string|null $previous
 * @property string|null $next
 * @property int $series_id
 * @property \Carbon\Carbon|null $published_at
 * @property \Carbon\Carbon|null $created_at
 * @property \Carbon\Carbon|null $updated_at
 */
```

Errors 12 i 13

Els errors que trobem amb el test de **ProfileInformationTest** es deuen al fet que **Livewire\Testable** ja no utilitza **$state** com una propietat accessible directament.

```
------ ------------------------------------------------------------------------------------
 Line   tests/Feature/ProfileInformationTest.php
------ ------------------------------------------------------------------------------------
 21     Access to an undefined property Livewire\Features\SupportTesting\Testable::$state.
        🪪  property.notFound
        💡 Learn more: https://phpstan.org/blog/solving-phpstan-access-to-undefined-property
 22     Access to an undefined property Livewire\Features\SupportTesting\Testable::$state.
        🪪  property.notFound
        💡 Learn more: https://phpstan.org/blog/solving-phpstan-access-to-undefined-property
------ ------------------------------------------------------------------------------------
```

Per a solucionar-ho, en comptes d'accedir directament a **$component->state**, utilitzare el mètode **get()** per obtenir l'estat actual del component.

```
1 Steven *
public function test_current_profile_information_is_available(): void
{
    $this->actingAs($user = User::factory()->create());

    $component = Livewire::test( name: UpdateProfileInformationForm::class);

    $state = $component->get('state'); // Obté l'estat del component

    $this->assertEquals($user->name, $state['name']);
    $this->assertEquals($user->email, $state['email']);
}
```

Errors finals

Tot i que no es una solució ideal, 12 o 13 errors aproximadament del nostre projecte estaven relacionats amb Eloquent, i amb aquesta configuració, millorem la qualitat del codi sense sacrificar la velocitat de desenvolupament. Aquesta estratègia permet gestionar millor els errors dinàmics sense generar conflictes amb l'ús estàndard d'Eloquent a Laravel.

```
includes:
    - vendor/larastan/larastan/extension.neon
    - vendor/nesbot/carbon/extension.neon

parameters:
    paths:
        - app/
        - routes/
        - tests/
    # Level 10 is the highest level
    level: 5

    ignoreErrors:
        # Suppress errors related to accessing undefined properties on Eloquent models
        - '#Access to an undefined property Illuminate\\Database\\Eloquent\\Model::.*#'

        # Suppress errors related to undefined methods on Eloquent models
        - '#Call to an undefined method Illuminate\\Database\\Eloquent\\Model::.*#'
#    ignoreErrors:
#        - '#PHPDoc tag @var#'
```

Un cop solucionats tots els errors en el nostre projecte podrem veure com s'executa l'analisis sense trobar cap error.

```
alumnat@steven:~/Documents/VideosAppSteven$ ./vendor/bin/phpstan analyse
Note: Using configuration file /home/alumnat/Documents/VideosAppSteven/phpstan.neon.
 57/57 [============================] 100%

 [OK] No errors
```