Tutorial 03-B to do in class – Remember to upload the repo link to Teams

DIP

Antes de iniciar:

- Terminar los tutoriales anteriores.
- Este tutorial muestra el ejemplo de una inversión de dependencias y de una inyección de dependencias.
- ¿Puede identificar donde está la inversión, y donde la inyección?
- Al final vemos un ejemplo de cómo sería sin inversión de dependencias.

A. An interface

Interface

- Go to *app/* create a folder *Interfaces*
- Go to *app/Interfaces/* create a file *ImageStorage.php* with the next content:

Add Entire Code

```
<?php

namespace App\Interfaces;
use Illuminate\Http\Request;

interface ImageStorage {
    public function store(Request $request): void;
}
```

B. A util library

Image local storage

- Go to *app/* create a folder *Util*
- Go to *app/Util/* create a file *ImageLocalStorage.php* with the next content:

Add Entire Code

```
<?php
```

```
namespace App\Util;

use App\Interfaces\ImageStorage;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Storage;

class ImageLocalStorage implements ImageStorage
{
    public function store(Request $request): void
    {
        if ($request->hasFile('profile_image')) {
            Storage::disk('public')->put(
                'test.png',
                file_get_contents($request->file('profile_image')->getRealPath())
            );
        }
    }
}
```

C. A service provider

Provider

- Go to *app/Providers/* create a file called *ImageServiceProvider.php* with the next content:

Add Entire Code

```
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use App\Interfaces\ImageStorage;
use App\Util\ImageLocalStorage;

class ImageServiceProvider extends ServiceProvider
{
```

```
    public function register(): void
    {
        $this->app->bind(ImageStorage::class, function (){
            return new ImageLocalStorage();
        });
    }
}
```

Registering the new provider

- In *bootstrap/providers.php*, make the following changes in **bold**.

Modify Bold Code

```
<?php

return [
    App\Providers\AppServiceProvider::class,
    App\Providers\ImageServiceProvider::class,
];
```

D.Rest of the code

Controller

- Go to *app/http/Controllers/* create a file *ImageController.php* with the next content:

Add Entire Code

```
<?php

namespace App\Http\Controllers;

use App\Interfaces\ImageStorage;
use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;
use Illuminate\View\View;

class ImageController extends Controller
```

```
{
    public function index(): View
    {
        return view('image.index');
    }

    public function save(Request $request): RedirectResponse
    {
        $storeInterface = app(ImageStorage::class);
        $storeInterface->store($request);

        return back();
    }
}
```

Routes

- Go to *routes/web.php* and add these new routes by adding the next lines at the end of the file (check **bold**).

Modify Bold Code

```
...
Route::get('/image', 'App\Http\Controllers\ImageController@index')->name("image.index");
Route::post('/image/save', 'App\Http\Controllers\ImageController@save')->name("image.save");
```

View

- Go to *resources/views/* create a folder *image*.
- Go to *resources/views/image/* create a file *index.blade.php* with the next content:

Add Entire Code

```
@extends('layouts.app')
@section("title", "Image Storage - DI")
@section('content')
<div class="container">
  <div class="row justify-content-center">
    <div class="col-md-8">
```

```
    <div class="card">
      <div class="card-header">Upload image</div>
      <div class="card-body">

      <form action="{{ route('image.save') }}" method="post" enctype="multipart/form-data">
        @csrf
        <div class="form-group">
          <label>Image:</label>
          <input type="file" name="profile_image" />
        </div>
        <button type="submit" class="btn btn-primary">Submit</button>
      </form>

      <img src="{{ URL::asset('storage/test.png') }}" />
      </div>
    </div>
  </div>
 </div>
</div>
@endsection
```

Artisan

- Run the next command in Terminal

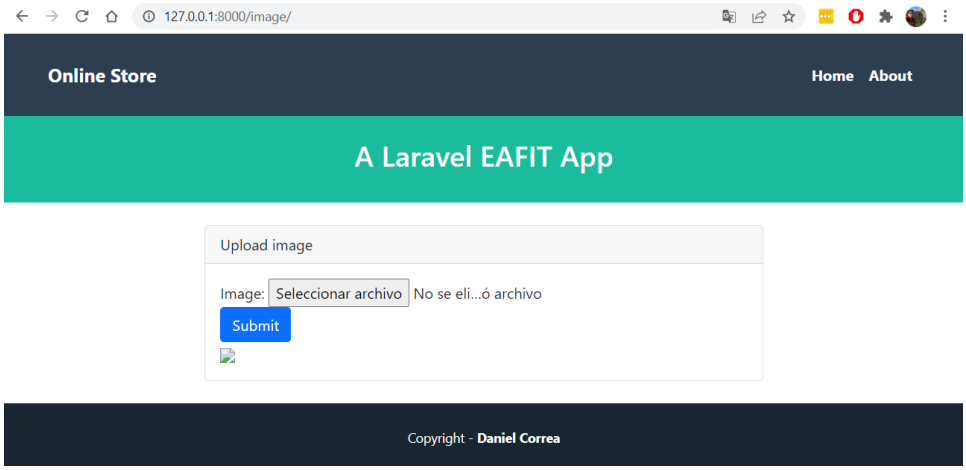
Execute in Terminal

```
php artisan storage:link
```

```
The [public/storage] directory has been linked.
```

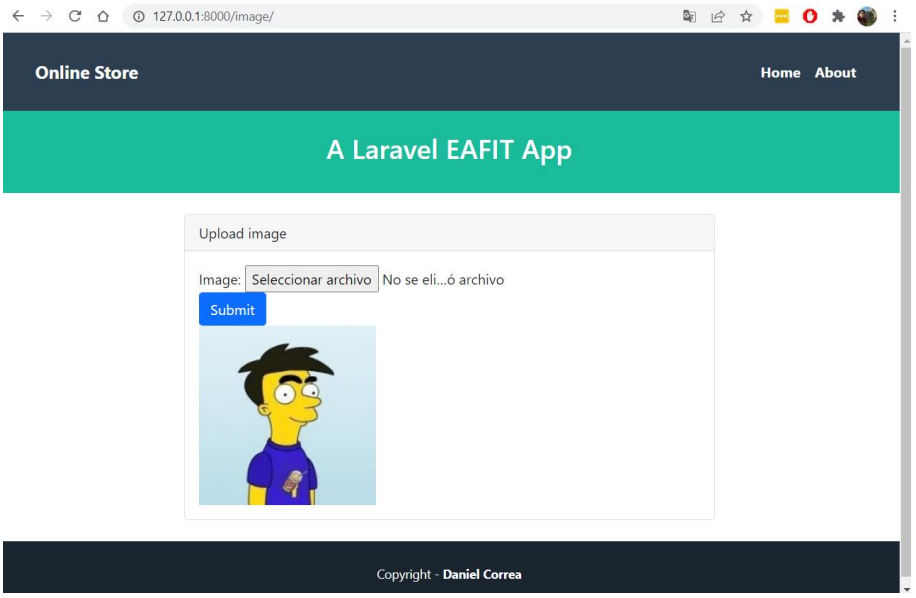
Execution

- Go to http://127.0.0.1:8000/image/



Upload image

- Upload a new image



SAME APPLICATION WITHOUT DEPENDENCY INVERSION

Controller

- Go to *app/http/Controllers/* create a file *ImageNotDIController.php* with the next content:

Add Entire Code

```
<?php
```

```
namespace App\Http\Controllers;

use App\Util\ImageLocalStorage;
use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;
use Illuminate\View\View;

class ImageNotDIController extends Controller
{
    public function index(): View
    {
        return view('imagenotdi.index');
    }

    public function save(Request $request): RedirectResponse
    {
        $storeImageLocal = new ImageLocalStorage();
        $storeImageLocal->store($request);

        return back();
    }
}
```

Routes

- Go to *routes/web.php* and add these new routes by adding the next lines at the end of the file (check **bold**).

Modify Bold Code

```
...
Route::get('/image-not-di', 'App\Http\Controllers\ImageNotDIController@index')-
>name("imagenotdi.index");
Route::post('/image-not-di/save', 'App\Http\Controllers\ImageNotDIController@save')-
>name("imagenotdi.save");
```

View

- Go to *resources/views/* create a folder *imagenotdi*.

- Go to *resources/views/imagenotdi/* create a file *index.blade.php* with the next content:

Add Entire Code

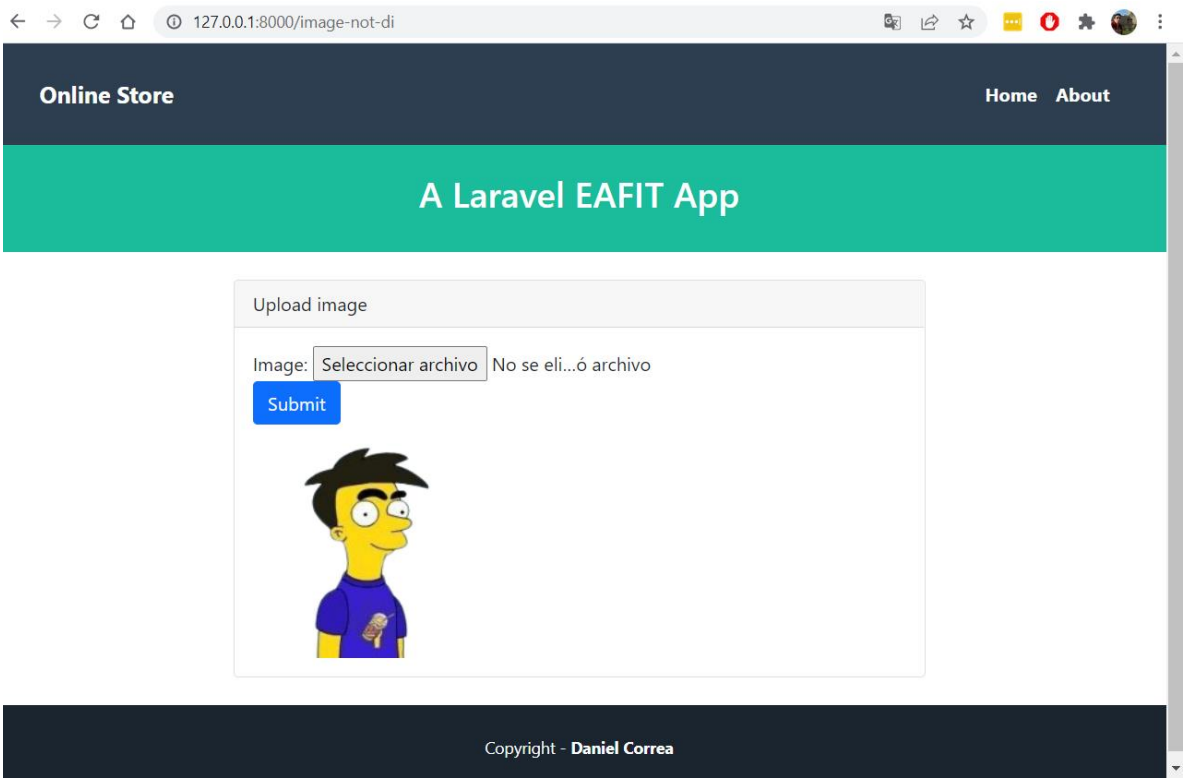
```
@extends('layouts.app')
@section("title", "Image Storage - DI")
@section('content')
<div class="container">
  <div class="row justify-content-center">
    <div class="col-md-8">
      <div class="card">
        <div class="card-header">Upload image</div>
        <div class="card-body">

        <form action="{{ route('imagenotdi.save') }}" method="post" enctype="multipart/form-data">
          @csrf
          <div class="form-group">
            <label>Image:</label>
            <input type="file" name="profile_image" />
          </div>
          <button type="submit" class="btn btn-primary">Submit</button>
        </form>

        <img src="{{ URL::asset('storage/test.png') }}" />
        </div>
      </div>
    </div>
  </div>
</div>
@endsection
```

Execution

- Go to http://127.0.0.1:8000/image-not-di



Preguntas

- ¿Puedes entender las diferencias entre las dos propuestas?
- ¿Ventajas/desventajas de cada una?
- Qué tal si tratas de comparar estas propuestas con lo siguiente (ver siguiente página):

Programación estructurada

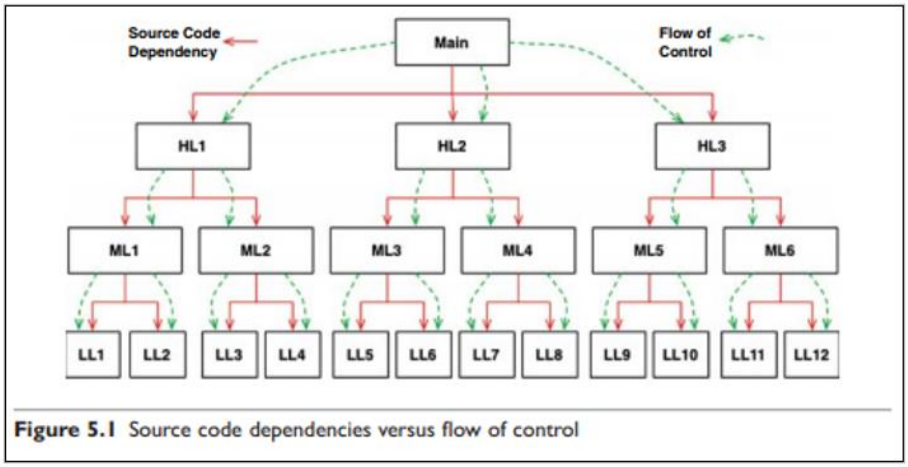
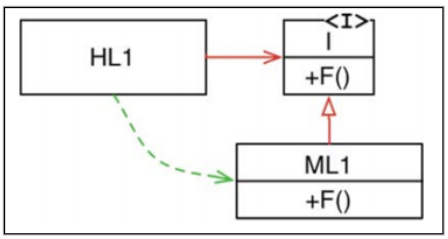


Figure 5.1 Source code dependencies versus flow of control

Podemos resumir el paradigma de programación estructurada de la siguiente manera:
La programación estructurada impone disciplina de transferencia directa de control.

Programación orientada a objetos

- En la figura de la derecha, el módulo "HL1" llama a la función F() en el módulo "ML1". El hecho de que llame a esta función a través de una interfaz es un posibilidad que brinda la POO. En tiempo de ejecución, la interfaz no existe. Y finalmente, "HL1" simplemente llama a F() dentro de "ML1".



- Por otro lado, veamos que la dependencia del código fuente (la relación de herencia) entre "ML1" y la interfaz "I" apunta en la dirección opuesta en comparación con el flujo de control. Esto se llama "**inversión de dependencias**", y sus implicaciones para el arquitecto de software son profundas.

Podemos resumir el paradigma de programación orientada a objetos de la siguiente manera:
La programación orientada a objetos impone disciplina de transferencia indirecta de control.

¿Será que con el ejemplo ya te queda clara esa carreta? ¿Sí? ¿No? La próxima clase conversaremos sobre esto