**CREAR UNA APLICACIÓN CON LARAVEL**

**¿Por qué TDD?**

TDD (o Test Driven Development) nos permite enfocarnos en la funcionalidad de la aplicación, dejando a un lado aspectos como el diseño que pueden abordarse una vez que “la prueba pase”, es decir que la funcionalidad básica esté lista.

**Módulos**

El curso lo presentaremos en tres partes, Back end, Front end y Comportamiento, y crearemos los siguientes módulos principales:

Categorías

Posts

Comentarios

Votos

Usuarios

Perfiles

Autenticación

**INSTALACIÓN Y CONFIGURACIÓN DE UN NUEVO PROYECTO CON LARAVEL 5.3**

Instalación y configuración del proyecto

Para crear la aplicación ejecuta:

composer create-project laravel/laravel foro "5.3.\*"

Crea una base de datos llamada foro, usando la herramienta que tengas disponible en tu entorno: phpMyAdmin, Sequel Pro, etc.

Instala el sistema de autenticación que viene por defecto en Laravel ejecutando:

php artisan make:auth

Modifica el archivo .env para configurar la variable de entorno para URL de la aplicación y las credenciales de la base de datos, es decir, edita las siguientes llaves:

APP\_URL=http://foro.app

DB\_DATABASE=foro

DB\_USERNAME=tu-username

DB\_PASSWORD=tu-password

Cambia el nombre de la aplicación en el archivo config/app.php modificando la clave 'name' => 'Foro'

Instala dos paquetes de terceros que usaremos en la aplicación:

styde/html

graham-campbell/markdown

**CONFIGURACIÓN PARA EL ENTORNO DE PRUEBAS AUTOMATIZADAS (PHPUNIT)**

En esta lección vamos a configurar la base de datos de prueba y vamos a preparar la estructura básica para agregar pruebas automatizadas a nuestra aplicación.

Es muy importante tener dos base de datos en nuestro proyecto local:

Una para las pruebas que realizaremos en el navegador: foro

Otra para las pruebas automatizadas: foro\_tests

Para ejecutar las migraciones que creen las tablas para la base de datos foro, ejecutamos:

php artisan migrate

Sin embargo, para ejecutar las migraciones en la base de datos foro\_tests debemos hacer algunas configuraciones:

Crear una nueva conexión de base de datos o DB\_CONNECTION en el archivo config/database.php que permitirá trabajar con la base de datos para las pruebas automatizadas:

'mysql\_tests' => [

'driver' => 'mysql',

'host' => env('DB\_HOST', 'localhost'),

'port' => env('DB\_PORT', '3306'),

'database' => env('DB\_DATABASE\_TESTS', 'foro\_tests'),

'username' => env('DB\_USERNAME', 'forge'),

'password' => env('DB\_PASSWORD', ''),

'charset' => 'utf8',

'collation' => 'utf8\_unicode\_ci',

'prefix' => '',

'strict' => true,

'engine' => null,

],

Es decir, que por defecto DB\_CONNECTION seguirá siendo mysql con la configuración de nuestra base de datos foro, pero cuando queremos trabajar con la base de datos foro\_tests nos referiremos a la conexión mysql\_tests. Así que para ejecutar las migraciones para esta base de datos, escribimos en consola:

php artisan migrate --database=mysql\_tests

También debemos asegurarnos que cada vez que se ejecuten las pruebas se haga con la base de datos correcta y esto lo logramos modificando el archivo phpunit.xml con el cual se sobrescriben algunas variables de entorno al momento de ejecutar las pruebas, es decir, agregamos:

<env name="DB\_CONNECTION" value="mysql\_tests"/>

De esta manera, Laravel tomará como base de datos para las pruebas a foro\_tests.

Ahora para ejecutar las pruebas automatizadas usamos el comando:

vendor/bin/phpunit

Para probar que se está usando la bases de datos correcta, vamos a modificar la prueba que viene por defecto en /tests/ExampleTest.php para ahora usar la ruta de /routes/api.php.

En este caso, la ruta /api/user requiere que el usuario esté conectado, es decir, que la ruta está siendo restringida con el middleware auth pasándole el parámetro api así middware('auth:api').

Así que debemos crear un usuario para trabajar en la prueba usando model factory de User que viene por defecto en Laravel y se encuentra en /database/factories/ModelFactory.php

Con el método actingAs de Laravel puedes hacer que un usuario creado en una prueba se conecte a la aplicación, este método acepta como segundo argumento opcional el driver sobre el cual se quiere conectar como por ejemplo:

public function testBasicExample()

{

$user = factory(App\User::class)->create([

'name' => 'Duilio Palacios',

]);

$this->actingAs($user, 'api');

$this->visit('/api/user')

->see('Duilio Palacios');

}

Importante: las pruebas automatizadas deben pasar o fallar siempre sin depender de factores externos o de otras pruebas y además deben ser rápidas y la mejor forma que hemos conseguido para esto es crear una base de datos de prueba con la estructura más actual de las migraciones pero sin datos, y antes de ejecutar cada prueba vamos a hacer “seed” de los datos estrictamente necesarios para la misma (usando model factories en caso de requerirlo). Por ejemplo imaginemos que un usuario puede editar un post, los datos necesarios en este caso son:

Crear un usuario que va a editar el post

Crear un post a editar.

Una vez que la prueba termine de ejecutarse (pase o falle) nosotros queremos eliminar el usuario y el post, de manera que cuando ejecutemos la prueba nuevamente no obtengamos un error inesperado como por ejemplo que el usuario ya existe.

Hay dos formas de lograr esto, usando uno de los 2 traits que Laravel nos provee para el desarrollo de pruebas:

Illuminate\Foundation\Testing\DatabaseMigrations con el cual se ejecutará nuevamente todas las migraciones con migrate:refresh cada vez que se inicie un método de una prueba

Illuminate\Foundation\Testing\DatabaseTransactions que hace uso de transacciones de base de tatos para eliminar los datos creados por la prueba, al finalizar ésta. Por supuesto, ésta es la opción más rápida.

**CREACIÓN Y CONFIGURACIÓN DE UN REPOSITORIO DE GIT PARA NUESTRO PROYECTO EN LARAVEL 5.3**

En esta lección vamos a crear un nuevo repositorio de Git tanto en local como en remoto usando GitHub y vamos a hacer nuestro primer commit. El control de versiones nos permitirá trabajar con nuestro proyecto de manera segura manteniendo un historial de todos los cambios que vayamos realizando en nuestro código, no sólo es una herramienta útil sino que para muchos es algo indispensable ya que podremos estar seguros de que nuestro código siempre estará protegido incluso si algo le llegara a pasar a nuestra computadora, así que vamos a ver cómo podemos crear nuestro repositorio y cómo podemos resguardarlo en un servidor remoto usando GitHub.

Iniciar con un repositorio en git es muy sencillo ya que basta con ir a la terminal y dirigirnos al directorio donde tenemos nuestro proyecto y una vez ahí debemos ejecutar el comando necesario para inicializar el repositorio de manera local:

cd /ruta/del/proyecto

# Inicializar el repositorio

git init

Esto nos va a generar una carpeta oculta llamada .git que va a contener la “base de datos” con todos los cambios que vayamos haciendo a nuestro código, la cual es increíblemente ligera, así que no tendrás que preocuparte por que ocupe mucho espacio en tu disco duro.

Ya que hayamos inicializado el repositorio vamos a poder empezar a rastrear todos los cambios que vayamos haciendo al código, sin embargo, primero debemos decirle a git cuáles son los archivos que queremos estar vigilando con el controlador de versiones. Para poder ver cómo se encuentra el status de nuestro repositorio y poder ver los archivos que se empezarán a rastrear podemos ejecutar el siguiente código en la consola:

git status

Esto mostrará todos los archivos en color rojo indicandome que aún no se encuentran dentro del área de stage, la cual es la que va a monitorear los cambios en el código, para agregar todos los archivos debemos ejecutar el siguiente comando:

git add .

Y seguido a esto debemos crear nuestro primer commit, que es una especie de checkpoint e indica que todo el código hasta el momento en que registramos el commit debe estar reguardado, el commit lo podemos crear con el siguiente comando:

git commit -m "Mensaje del commit"

Aquí nosotros debemos indicar un mensaje indicando el motivo de este commit por lo que debe ser algo que describa la funcionalidad del código o el motivo del por qué se hizo ese commit.

Con esto ya tenemos todos los archivos en nuestro repositorio de git, pero hay ocasiones en las que deseamos excluir algunos archivos o directorios de nuestro repositorio, tales como son la carpeta node\_modules y vendor en nuestros proyectos de Laravel, afortunadamente esto ya viene pre-configurado con un archivo .gitignore ya que dentro de él se listan todos los archivos y/o directorios a los cuales no vamos a estar monitoreando con git.

Lo siguiente que debemos hacer es resguardar nuestros commits en un repositorio remoto, ya que es ahí donde podemos hacer que otras personas puedan colaborar con nosotros en el caso de que así lo desearamos, para esto debemos contar con al menos un commit y si queremos revisar nuestro historial para saber cuáles son nuestros commits podemos ejecutar el siguiente comando:

git log

Esto nos va a mostrar un listado detallado de todos los commits que tenemos en nuestro repositorio.

Ahora debemos crear un repositorio remoto, para esto tenemos una serie de opciones como lo son GitHub y Bitbucket, una vez que lo tengamos listo debemos enlazarlo al repositorio local, esto lo hacemos tecleando el siguiente comando.

#Crear enlace con el repositorio remoto

git remote add origin [git@github.com:usuario/proyecto.git](mailto:git@github.com:usuario/proyecto.git)

Esto creará el puente necesario para poder mandar todo lo que tenemos en nuestro repositorio local hacia GitHub, para hacerlo debemos usar el siguiente comando:

git push -u origin master

Opcionalmente podemos crear una rama o branch, esto es una copia fiel del repositorio en la cual podemos hacer cambios al código sin afectar nuestra rama base, para poder crearla debemos ejecutar el siguiente comando:

git checkout -b "nombre\_del\_branch"

Al crear la rama debemos también hacer un push de esta rama hacia nuestro repositorio remoto para poder mantener este branch asegurado, para hacerlo debemos hacerlo igual que lo hicimos anteriormente, con la única diferencia de que en lugar de utilizar nuestro branch master (que es nuestro branch principal) vamos a indicar el branch que acabamos de crear:

git push -u origin nombre\_del\_branch