https://doi.org/10.22201/dgtic.ctud.2024.2.3.61

Vol. 2, Núm. 3. julio-septiembre 2024, págs. 40 - 49

Automatización de tareas para desarrolladores de software en Laravel, por medio de comandos personalizados

Información del reporte:

Para citar este reporte técnico:

Sánchez Montes de Oca, P. Z. (2024) Automatización de tareas para desarrolladores de software en Laravel, por medio de comandos personalizados. Cuadernos Técnicos Universitarios de la DGTIC, *2* (3) páginas(40 - 49).

https://doi.org/10.22201/dgtic.ctud.2024.2.3.61

Paulina Zulen Sánchez Montes de Oca

Dirección General de Cómputo y de Tecnologías de Información y Comunicación

Universidad Nacional Autónoma de México

pzulen28@gmail.com

ORCID: 0009-0006-8638-8852

Resumen

En la creación de un producto de software, es común encontrarse con fragmentos de código similares en varias partes del sistema. A menudo, estos fragmentos sólo varían en los nombres de los componentes, lo que lleva a la práctica de copiar y pegar el código, para adaptarlo luego al nuevo componente. Con la finalidad de abordar esta problemática, surge la necesidad de crear comandos personalizados que sirvan como plantillas base para el desarrollo del sistema. Esto conlleva numerosos beneficios, como la reducción en tiempos en la creación de un componente, en el uso de recursos del sistema, una mayor eficiencia en el desarrollo del código, la estandarización (al utilizar un modelo específico para cada componente) y la capacidad de incorporar código de acuerdo con las necesidades individuales.

Palabras clave:

Laravel, comandos personalizados, estandarización, plantilla base

https://doi.org/10.22201/dgtic.ctud.2024.2.3.61 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 41 - 49

1. INTRODUCCIÓN

Durante el desarrollo de un sistema de software, la selección de las herramientas que mejor se ajusten a las necesidades y requisitos del proyecto es un factor crítico de éxito. En la Dirección General de Cómputo y de Tecnologías de Información y Ccomunicación se han entregado más de 15 productos de software utilizando el *framework* Laravel. De acuerdo con Morales (2023), "Laravel es un *framework* de desarrollo web de código abierto escrito en PHP que se utiliza para crear aplicaciones web de alta calidad y escalables, siendo uno de los *frameworks* de PHP más populares".

La elección de Laravel se basa en su código legible, que facilita su comprensión, así como su eficiente manejo de bases de datos a través de Eloquent ORM (Mapeo Objeto-Relacional). Además cuenta con características robustas, como la adopción del patrón de arquitectura Modelo-Vista-Controlador (MVC), el cual, según Zapata (2023), "con esta arquitectura es posible separar la aplicación en 3 capas: en la capa Modelo se muestran los datos del programa; la capa Vista presenta la interfaz gráfica del software; y la capa Controlador que responde a las acciones del usuario y envía peticiones a la capa Modelo cuando el usuario solicita cierta información". Adicionalmente, su consola integrada Artisan permite agilizar la automatización de tareas mediante comandos de línea.

Más allá de simplemente utilizar los comandos proporcionados por Laravel, este reporte tiene como objetivo brindar recomendaciones para el desarrollo de comandos personalizados, adaptándolos a un sistema base, al minimizar el tiempo de creación de componentes y aprovechar las características del *framework*, para crear sistemas escalables, fáciles de mantener y con una alta reutilización de código.

Para la creación de estos comandos personalizados y con base en su compatibilidad con PHP, se han empleado funciones para definir las variables utilizadas en el desarrollo de estos comandos, al proporcionar una plantilla base para los diferentes componentes del sistema y proyectos futuros. Esto garantiza características homogéneas y una estructura uniforme en las clases y funciones, lo que permite reducir el tiempo de desarrollo. Además, al estar optimizados y estandarizados, permiten a los desarrolladores enfocarse en características y desafíos específicos de cada sistema en desarrollo.

2. DESARROLLO TÉCNICO

Se realizó una instalación de Laravel y el desarrollo de un comando personalizado, que proporcionó detalles sobre su ubicación y estructura. Este software ofrece varios comandos esenciales, entre ellos:

- *Make:* Genera una plantilla de clase, que proporciona un esquema donde se define el comportamiento y sus propiedades.
- *Migrate:* Se utiliza para crear tablas de migración en la base de datos e identifica las migraciones previamente ejecutadas para evitar duplicidades.
- *Sail*: Una herramienta que facilita la creación de entornos de desarrollo basados en Docker (tecnología de contenedores), que permite la rápida configuración de aplicaciones Laravel.
- *Route*: Define las rutas para una aplicación Laravel, estableciendo la conexión entre las solicitudes HTTP y la lógica de la aplicación.

https://doi.org/10.22201/dgtic.ctud.2024.2.3.61 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 42 - 49

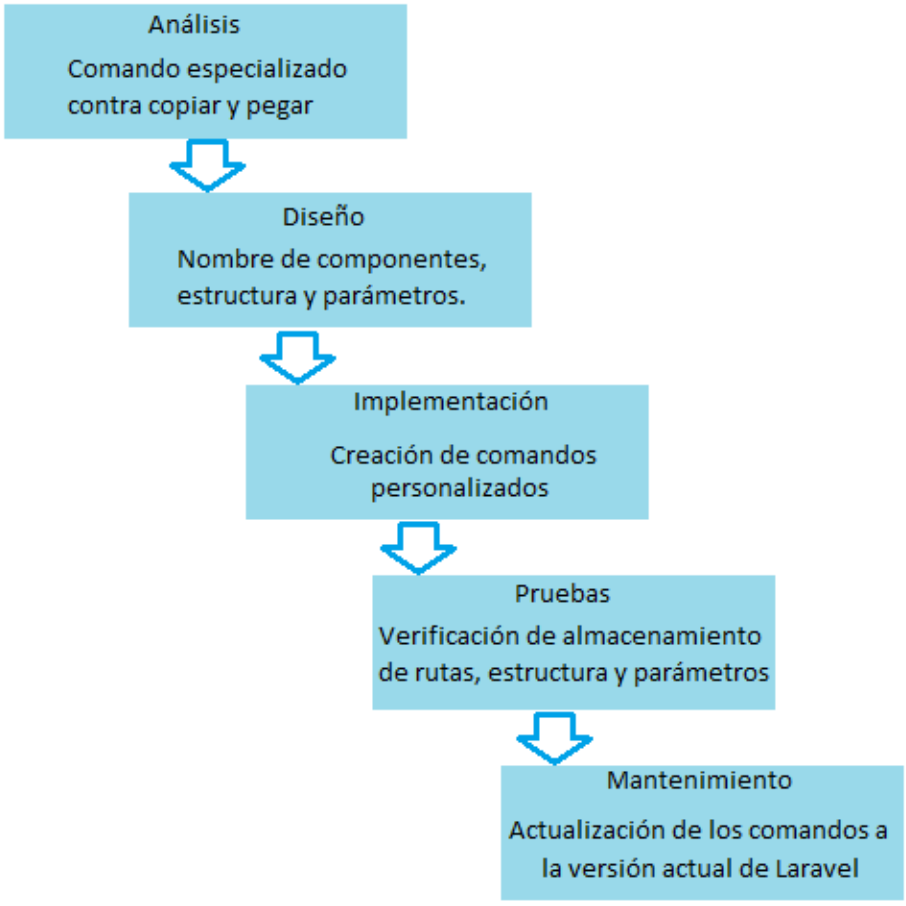
Si se escribe "php artisan" en la terminal del editor de código se despliega una lista completa de comandos disponibles, junto con sus descripciones. Al crear comandos personalizados, estos se agregan a la lista, lo que permite que los miembros del equipo de desarrollo los utilicen. Se eligió Visual Studio Code como editor de código, debido a que "la instalación de extensiones permite un desarrollo fluido, también integra herramientas como la terminal que se usará mucho para hacer toda instalación de paquetes y ejecutar comandos para levantar el mismo proyecto" (Esquivel, 2023).

2.1 METODOLOGÍA

La metodología empleada para desarrollar estos comandos personalizados es el modelo de cascada (Figura 1), que consta de las etapas de análisis, diseño, implementación, pruebas y mantenimiento. Según Universitat Carlemany (2024), esta metodología "permite organizar el trabajo en vertical, de arriba a abajo. Esto significa que se realiza una actividad por fases secuenciales y que no es posible pasar a la siguiente hasta que no se haya verificado la anterior".

Figura 1

Modelo de cascada



https://doi.org/10.22201/dgtic.ctud.2024.2.3.61 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 43 - 49

Análisis

Dadas las características de los sistemas web, se identificó que varios de sus componentes contenían código repetitivo, lo que resultaba en una inversión de tiempo innecesaria al reescribirlo. Además, cada desarrollador empleaba una nomenclatura diferente, lo que causaba errores de programación y prolongaba los tiempos de entrega, incrementando los costos. Tras una exploración, se descubrió que Laravel ofrece la funcionalidad de crear comandos personalizados que permiten generar plantillas para la reutilización de código. Se optó por esta alternativa debido a su facilidad de uso y a la documentación disponible en la página oficial de Laravel; y se creó el primer comando personalizado. La Tabla 1 muestra la comparación de utilizar un comando personalizado y la acción copiar – pegar (forma tradicional).

Tabla 1

Comparación entre los comandos personalizados y la práctica de copiar y pegar código

	Comandos personalizados	**Copiar y Pegar Código (forma tradicional)**
Eficiencia y Productividad	Automatización: Genera automáticamente componentes. Consistencia: Disminuye el riesgo de errores humanos. Escalabilidad: Aumenta la productividad del equipo. Tiempo: Máximo un minuto.	Manualidad: Los desarrolladores deben copiar y ajustar manualmente el código. Inconsistencias: Incrementa la probabilidad de errores. Tiempo: Seleccionar, copiar, pegar y modificar el código.
Mantenimiento	Facilidad de mantenimiento: Los cambios se realizan en el comando personalizado.	Dificultad de mantenimiento: Cada instancia del código copiado debe cambiarse manualmente.
Estandarización	Estandarización: Código estandarizado y uniforme.	Variabilidad: Cada desarrollador puede implementar variaciones.
Adaptabilidad	Adaptabilidad: Facilita la personalización de componentes.	Rigidez: Cada adaptación debe hacerse manualmente.
Flexibilidad	Flexibilidad: Permite la inclusión de lógica adicional o validaciones específicas.	Mayor complejidad: Adaptar el código copiado a diferentes contextos puede introducir complejidad y errores.

Diseño

En esta fase se identificaron las rutas donde se almacenaron los diferentes comandos personalizados, y se definió la nomenclatura de los nombres de cada comando consultables en la Figura 2, así como el nombre con el cual se ejecutan, y la estructura de sus parámetros (Figura 3).

https://doi.org/10.22201/dgtic.ctud.2024.2.3.61 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 44 - 49

Figura 2

Nombre de ejecución y parámetros de comandos personalizados

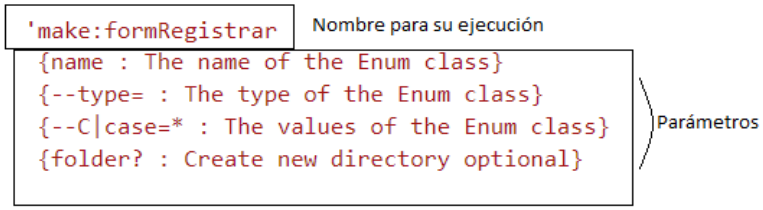
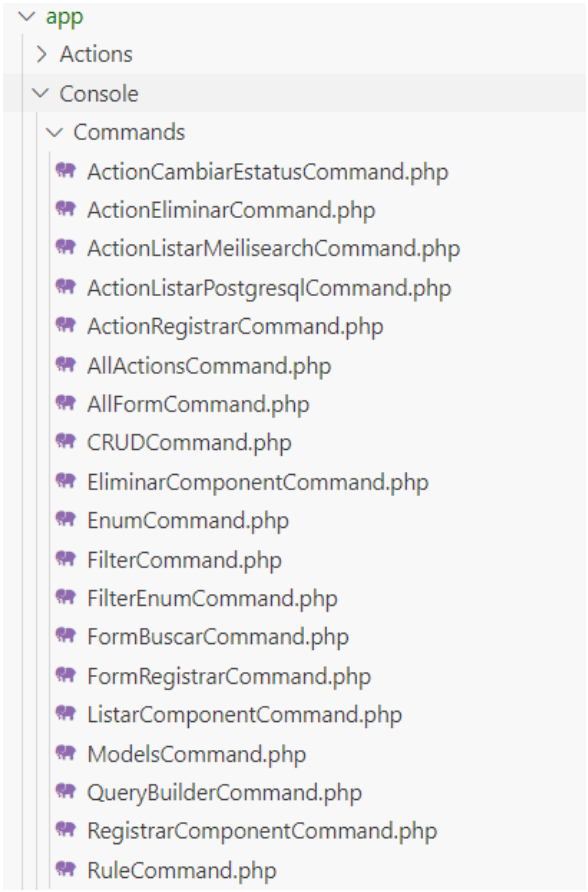


Figura 3

Ruta y nomenclatura de comandos personalizados



https://doi.org/10.22201/dgtic.ctud.2024.2.3.61 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 45 - 49

Implementación

Esta fase inició con el establecimiento de la estructura general de un sistema base en Laravel, mediante la creación de un nuevo sistema con la instalación de *Composer*, que según Ávalos et al. (2023), "es un manejador de dependencias para proyectos del lenguaje de programación PHP", lo que permite declarar las bibliotecas necesarias para la estandarización y actualización del sistema, de forma eficiente.

El proceso inició con la descarga e instalación de la última versión de PHP Composer. Luego, se creó un nuevo proyecto Laravel por medio del comando *composer create-project laravel/laravel proyecto-base-laravel*, siguiendo los lineamientos establecidos en la documentación de Laravel para visualizar el proyecto en el navegador en http://localhost.

Para crear la estructura base del comando personalizado, se utilizó el comando *php artisan make:command EjemploCommand*, y se asignó un nombre que invoca el comando personalizado, estructurado por el nombre del componente, su acción y una breve descripción (Laravel, 2011-2024).

Asimismo, en la función *handle* se emplearon funciones PHP para estructurar las variables necesarias para la creación de rutas, *namespaces,* nombres y validaciones de los componentes, como se muestra en la siguiente tabla:

Tabla 2

Funciones de Laravel y PHP

Funciones utilizadas de Laravel	Funciones utilizadas de PHP
Str::plural(): Ayuda a que el dato entrante en la terminal siempre lo regrese en plural.	ucfirst(): Convierte el primer carácter de la cadena ingresada a mayúsculas.
Str:replace(): Reemplaza la doble diagonal en caso de no haber ingresado una carpeta adicional, además de cambiar la diagonal por diagonal invertida.	file_exists: Verifica si existe un fichero o directorio.
Str:of)->between: Regresa la cadena que se encuentra entre los valores dados.	is_dir: Indica si el nombre de archivo es un directorio.
Str:afterLast(): Devuelve todo lo que ocurre después de la última aparición del valor dado en una cadena.	file_put_contents(): Escribir datos en un archivo.
Str::before: Retorna todo lo que está antes del valor dado en una cadena.	php_array_map(): Aplica la devolución de llamada a los elementos de las matrices dadas.
Str::snake() Para que devuelva la cadena separada por guiones bajos.	implode(); Une elementos de un array en un string
Str::headline() Convierte cadenas delimitadas por mayúsculas, guiones o guiones bajos en una cadena delimitada por espacios con la primera letra de cada palabra en mayúsculas.	N/A

Se creó una función de tipo *get* para generar la plantilla, para asegurar que devuelva el código estandarizado.

https://doi.org/10.22201/dgtic.ctud.2024.2.3.61 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 46 - 49

Figura 4

Función getFormBuscar

```php
protected function getFormBuscar()
{
    $cases = $this->option('case');
    $cases = array_map(function($case) {
        return "public $".Str::snake($case)." = '';";
    }, $cases);

    $attributes = $this->option('case');
    $attributes = array_map(function($attributes) {
        return "'".Str::snake($attributes)."' => '".Str::headline($attributes)."',";
    }, $attributes);

    $rules = $this->option('case');
    $rules = array_map(function($rules) {
        return "'".Str::snake($rules)."' => ['nullable', 'string'],";
    }, $rules);

    $isDirty = $this->option('case');
    $isDirty = array_map(function($isDirty) {
        return "$"."this"."->".Str::snake($isDirty)."";
    }, $isDirty);

    return '<?php
```

Figura 5

Plantilla getFormBuscar

```
namespace [namespace];

use Livewire\Attributes\Validate;
use App\Livewire\Forms;

class [classComponent] extends Form
{
    ' . implode("\n    ", $cases) . '

    public $validationAttributes = [
    ' . implode("\n    ", $attributes) . '

    ];

    public function rules(): array
    {
        return [
    ' . implode("\n    ", $rules) . '
        ];
    }

    public function isDirty(): bool
    {
        return ' . implode("\n            || ", $isDirty) . '
        ;
    }
}
';
    }
}
```

https://doi.org/10.22201/dgtic.ctud.2024.2.3.61 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 47 - 49

Pruebas

Se verificó que el componente creado se almacenara en la ruta adecuada, que contuviera las funciones y la estructura principal según el tipo de componente, y que cumpliera con la nomenclatura estipulada en la fase de análisis. Además, se comprobó que inyectara automáticamente todas las dependencias definidas en el método *handle*, para asegurar su adecuado funcionamiento.

Mantenimiento

Al ser una plantilla, ésta se ha adaptado de acuerdo con el tipo de componente creado y la versión actual de Laravel, ya que al existir cambios en la documentación, varía la estructura de algunos componentes.

2.2 RESULTADOS

Una vez creada la plantilla, se escribe en la terminal el nombre del comando, nombre de la carpeta del componente, los casos que se crearán con la bandera –case = nombreCaso, y la carpeta padre, la cual es opcional, y se verá en la ruta asignada por el grupo de desarrollo, como se muestra en el ejemplo de la Figura 6.

Figura 6

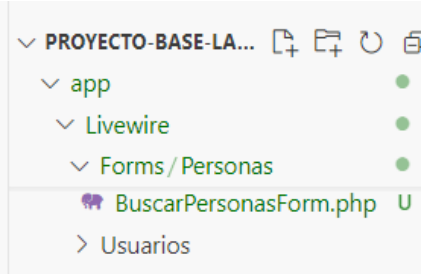
Creación del Form Request por la terminal de comandos BuscarPersonasForm

```
pzule@DESKTOP-NM6U7A3:~/proyecto-base-laravel-10$ sail artisan make:formBuscar personas --case=apellidoPaterno --case=apellidoMaterno
--case=correoElectronico --case=rol Forms
Form buscar created successfully.
pzule@DESKTOP-NM6U7A3:~/proyecto-base-laravel-10$
```

La ruta donde se almacena el *Form Request* se visualiza en el explorador del editor de código, como se ve en el ejemplo de la Figura 7.

Figura 7

Ruta del Form Request



Al seleccionar el *Form Request* "BuscarPersonasForm.php" se muestra el componente creado.

Se obtuvo un *Form Request* con una estructura base, donde el desarrollador únicamente adaptó las validaciones de acuerdo con los requerimientos específicos del *Form Request*.

Para el desarrollo de un *Form Request* de la forma tradicional, el desarrollador deberá ingresar a la terminal del editor y escribir *sail php artisan livewire:form BuscarAreasForm*, si utiliza *docker*, o en su defecto omitir

https://doi.org/10.22201/dgtic.ctud.2024.2.3.61 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 48 - 49

sail. El sistema muestra una plantilla básica con el *namespace* (ruta del *form* creado), las importaciones de la clase que permiten crear las instancias del *form* y la clase con el nombre ingresado.

Posteriormente, el desarrollador deberá crear, copiar y pegar manualmente los atributos públicos, protegidos o privados según las necesidades, para crear las validaciones de esos atributos y las reglas de validación. En el ejemplo visto se crea una función adicional llamada *isDirty*, para determinar si alguno de los atributos del formulario ha sido modificado respecto a su valor inicial (Figura 9).

Figura 9

Form Request BuscarAreasForm

app > Livewire > Forms > BuscarAreasForm.php

```php
<?php

namespace App\Livewire\Forms;

use Livewire\Attributes\Validate;
use Livewire\Form;

class BuscarAreasForm extends Form
{
    //
}
```

Al ser un ejemplo básico, la diferencia de tiempo entre crear un *form* manual a un *form* con un comando personalizado es de 15 minutos por cada uno; pero al poder crear, como se ha mencionado antes, diferentes tipos de comandos personalizados y éstos poder ser reutilizados, el tiempo se incrementa considerablemente, dependiendo de la cantidad de componentes necesarios en un sistema.

Crear comandos personalizados es una práctica mucho más eficiente, segura y profesional, en contraste con el simple copiado y pegado de código. Los comandos personalizados mejoran la productividad, aseguran la consistencia y estandarización del código, y facilitan el mantenimiento y la actualización del software. Por otro lado, copiar y pegar código puede ser rápido en el corto plazo, pero a largo plazo, introduce riesgos significativos de errores, inconsistencias y dificultades de mantenimiento.

CONCLUSIONES

El uso de comandos ha facilitado a los desarrolladores la creación de componentes básicos, así como la reducción de los tiempos de desarrollo. Además, la personalización de estos comandos permite contar con información estandarizada. Esto mejora la legibilidad del código, facilita el mantenimiento, reduce

https://doi.org/10.22201/dgtic.ctud.2024.2.3.61 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 49 - 49

errores y asegura que los componentes compartan las mismas características. A su vez, cada componente puede incorporar su propio código sin alterar la estructura base del sistema.

En el ejemplo práctico presentado, una vez creado un comando personalizado, solamente se necesita invocar el nombre del comando, el nombre del componente y los casos de validación deseados. Esto simplifica el proceso a una sola línea de código ingresada en la terminal de Artisan, creando así un *Form Request* en este caso, pero se pueden crear diversos tipos de componentes y vistas, como listados, registros, actualizaciones, o comandos que gestionen el sistema internamente, incluyendo *enums, filters, forms, models,* y *rules*, entre otros.

La aplicación de esta metodología no sólo optimiza el desarrollo, sino que también fomenta la estandarización y la eficiencia en la gestión de proyectos de software, al ofrecer un enfoque sistemático y adaptable para diversas necesidades de desarrollo.

REFERENCIAS

Avalos Rojas, R., Diaz Contreras, S., Valdez Ramos, M., Javier Baeza, J. (2023, Enero - Marzo). Revisión de Framework Laravel y su aplicación en sistema web de mesa de ayuda. Innovación y Desarrollo Tecnológico Revista Digital. 149 - 160 Recuperado de https://iydt.files.wordpress.com/2022/12/1_17_revision-de-framework-laravel-y-su-aplicacion-en-sistema-web-de-mesa-de-ayuda.pdf

Esquivel Treviño, C., Martínez Moreno, M., Garduño Gaffare, M. P., Moreno Ramírez, R. E., & Ruíz Jiménez , J. (2023). Impacto de la utilización de un Framework como Laravel en el desarrollo de un sitio web para servicios de modificaciones corporales. Ciencia Latina Revista Científica Multidisciplinar, 7(3), 3217-3236. https://doi.org/10.37811/cl_rcm.v7i3.6402

Laravel (2011 - 2024). Recuperado de https://laravel.com/docs/11.x

Laravel - Artisan Console. (2011 – 2024) Recuperado de https://laravel.com/docs/11.x/artisan#main-content

Manual de PHP. (2001-2024). Recuperado de https://www.php.net/manual/es/

Universitat Carlemany (Marzo, 2024). Metodologías de desarrollo de software. Recuperado de https://www.universitatcarlemany.com/actualidad/blog/metodologias-de-desarrollo-de-software/

Zapata – Vega, S., Tejada Castro, M. I., Samaniego Orrala, J. A., Peñafiel Cox, M. F., & Guerrero –Zambrano, E. (2023). Desarrollo de una plataforma digital para la lectura autónoma del idioma inglés mediante la técnica de comprensión literal. Revista De La Universidad Del Zulia, 14(41), 555-566. https://doi.org/10.46925//rdluz.41.31