

**PRÁCTICA FINAL LARAVEL. 2ª EVALUACIÓN.**

**DESARROLLO WEB EN ENTORNO SERVIDOR.**

**Juan Antonio Dólera Jiménez**

**2º Desarrollo de Aplicaciones Web**

**Profesor/a: Fernando Ureña**

Contenido

[Como desplegar la aplicación 3](#_Toc189426899)

[Sprint 1: Configuración inicial del proyecto y autenticación 5](#_Toc189426900)

[Sprint 2: Gestión de categorías y productos 16](#_Toc189426901)

# Como desplegar la aplicación

Para desplegar la aplicación en nuestro equipo necesitamos lo siguiente:

1. **Descargar e instalar Xampp.**

Para instalar este software debemos dirigirnos a la siguiente url:

<https://www.apachefriends.org/es/index.html>

Y hacer click en el botón de la pantalla, después, debemos seguir la instalación siguiendo las instrucciones que nos indica.

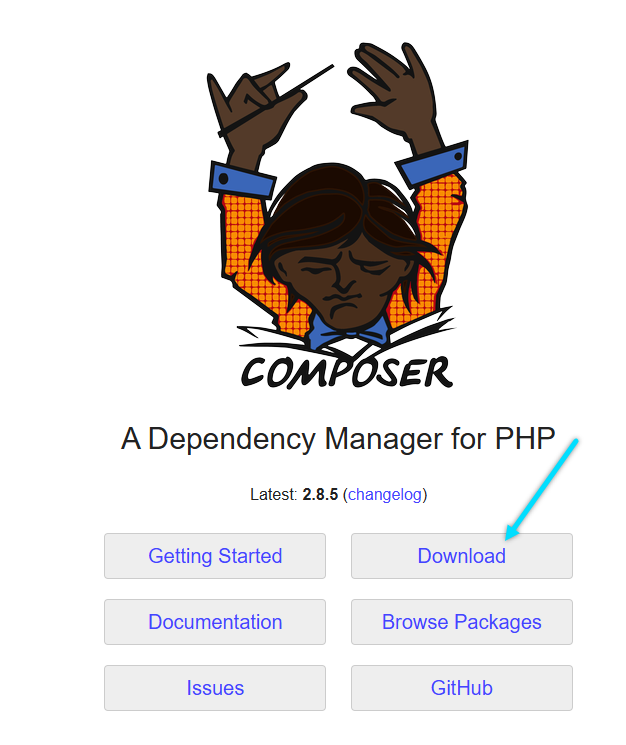


1. **Descargar e instalar Composer**

Para instalar este software debemos dirigirnos a la siguiente url:

<https://getcomposer.org/>

Y hacer click en el botón de la pantalla, después, debemos seguir la instalación siguiendo las instrucciones que nos indica.

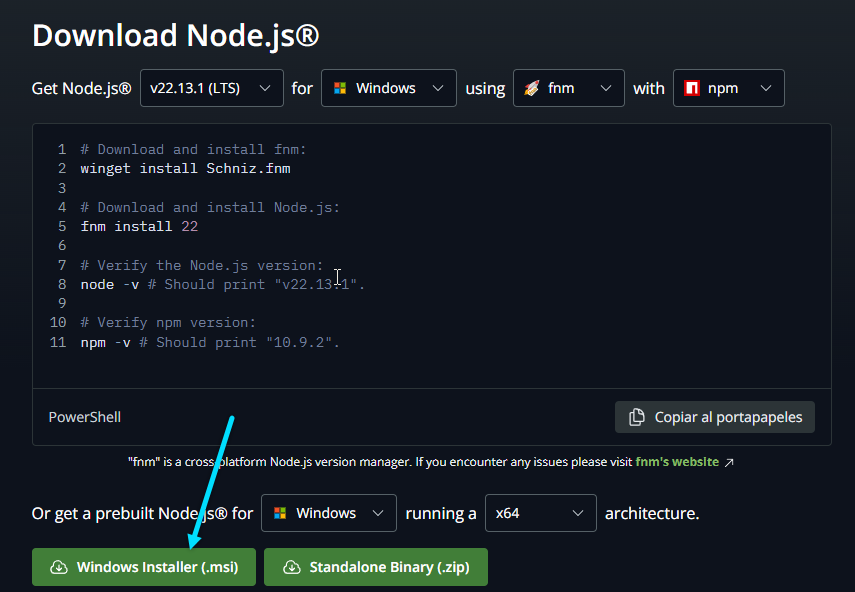


1. **Descargar e instalar Node.js**

Para instalar este software debemos dirigirnos a la siguiente url:

<https://nodejs.org/es/download>

Y hacer click en el botón de la pantalla, después, debemos seguir la instalación siguiendo las instrucciones que nos indica.



1. **Descargar o clonar el repositorio**

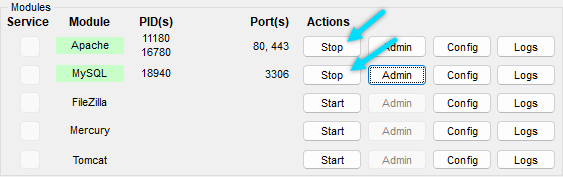
Para descargar el repositorio, debemos dirigirnos a la siguiente url de github:

<https://github.com/juanantonioVM/restaurant.git>

Una vez con la url, podemos clonar el repositorio a través de softwares de seguimiento de proyectos como Fork, pulsando sobre Clone Repository, colocando la url del repositorio y eligiendo la ruta de destino de donde lo queremos clonar.

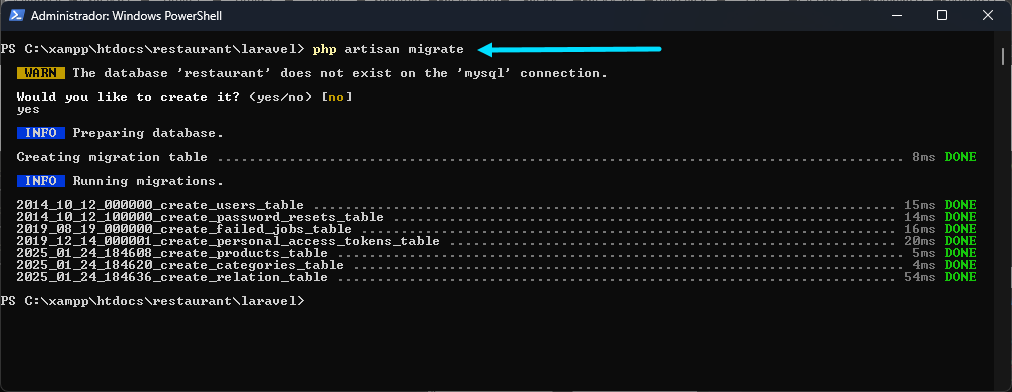
También podemos descargar el proyecto descargando el proyecto en formato comprimido desde github.

Para desplegar la aplicación debemos colocar este proyecto dentro de la carpeta htdocs de Xampp, y, una vez ahí, abrir este programa para iniciar el servidor en local, pulsando en los siguientes botones:

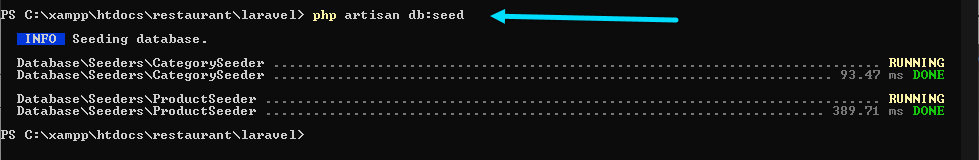


1. **Realizar las migraciones y seeders**

Para realizar las migraciones, debemos abrir nuestro proyecto desde la PowerShell de Windows en modo administrador, y colocar el siguiente comando:



Como vemos, confirmando la pregunta que nos realiza, nos crea también la base de datos sobre la que se va a apoyar nuestro proyecto. Ahora debemos implementar los seeders, para los cuales usaremos el siguiente comando, con el cual se nos transferirán todos los datos:



Una vez puesto, solo tendremos que registrarnos con el sistema que nos proporciona Laravel UI, y ya podremos navegar por nuestra página con normalidad. Tendremos que entrar las siguientes url’s para ver las páginas principales de nuestro proyecto y comenzar a usarlo:

Vista de la carta: <http://localhost/restaurant/laravel/public/menu>

Vista del administrador: <http://localhost/restaurant/laravel/public/login>

# Sprint 1: Configuración inicial del proyecto y autenticación

**Objetivo**: Configurar el entorno, establecer la arquitectura básica (MVC), y

preparar el sistema de autenticación y control de versiones.

**Tareas:**

1. **Configuración inicial:**

* Crear el proyecto Laravel.

Para crear el proyecto con laravel, debemos escribir la siguiente línea de comandos en la PowerShell, dentro de la carpeta de proyecto que queremos empezar.

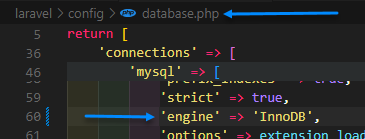


* Configurar base de datos (MySQL) y archivo .env.

Dentro del archivo .env, cambiamos esta línea de código para asignar el nombre de la base de datos.



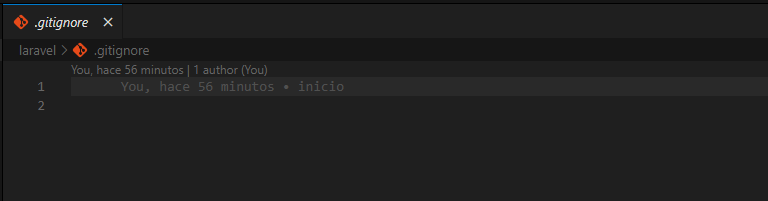
Más adelante, cuando hagamos las migraciones de las tablas desde la PowerShell se creará automáticamente la base de datos en PHPMyAdmin. Para evitar fallos con las migraciones, debemos cambiar esta línea en el siguiente archivo:



* Configurar un sistema de control de versiones con Git y establecer un repositorio en GitHub.

Para unir nuestro repositorio local con GitHub hemos usado Fork. Hemos creado un nuevo repositorio en la web de GitHub, y con Fork hemos iniciado el repositorio en local. Para subirlo a GitHub simplemente debemos hacer un push con el mismo Fork e implementar el enlace del repositorio en la nube, el cual se nos crea cuando hacemos el repositorio.

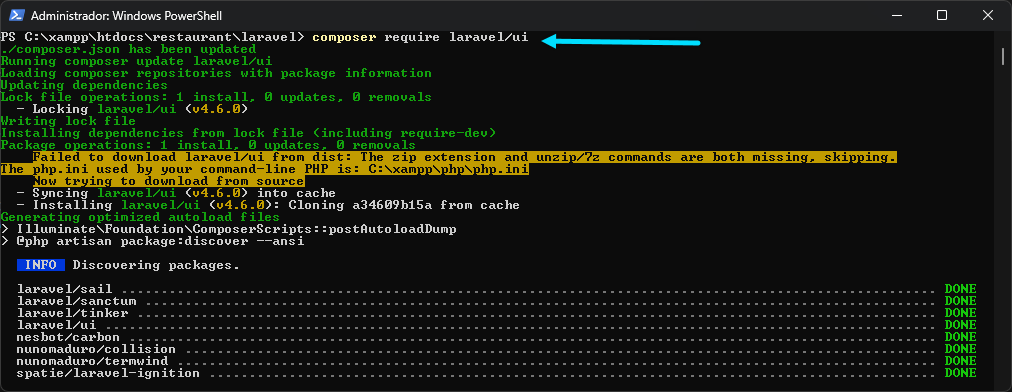
Antes de subirlo a GitHub, debemos borrar el archivo de .gitignore para que no haya ningún archivo que perdamos cuando hagamos el push. Este archivo lo encontramos en la carpeta de laravel. En mi caso lo he dejado vacío.

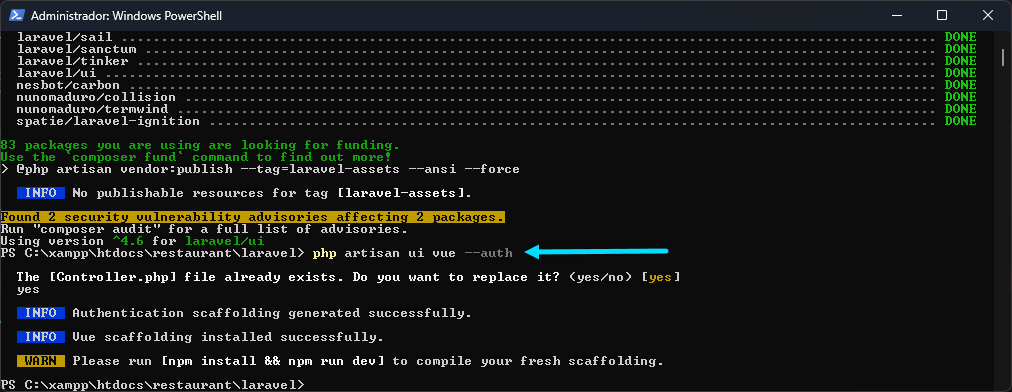


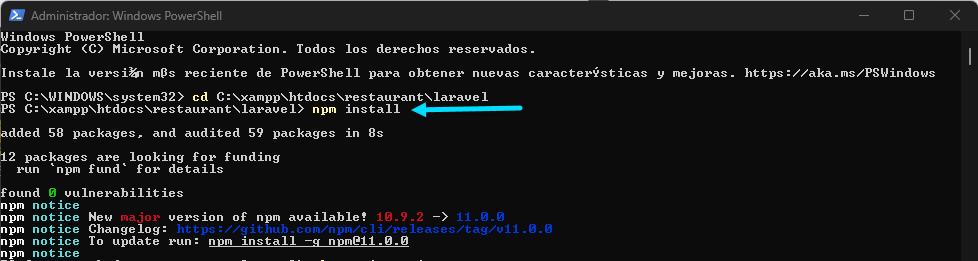
1. **Autenticación:**

* Usar Laravel Breeze o Laravel UI para generar un sistema básico de autenticación.

Para crear el sistema de autenticación se han ejecutado los siguientes comandos en la PowerShell de Windows, en modo administrador:



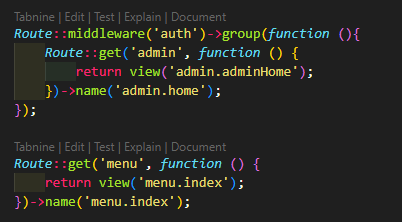




Una vez instalado todo esto, se nos crea unos archivos dentro de nuestro proyecto con los que podemos configurar toda la parte de la seguridad de la aplicación.

* Configurar acceso a parte pública y privada.

Para configurar el acceso a las diferentes partes de la aplicación se ha modificado el archivo de rutas web.php, para dejarlo de la siguiente manera:



Dentro de la ruta middleware, se incluyen todas las zonas de la aplicación que no queremos que entre nadie que no se encuentre registrado en la aplicación. En este caso, la vista del administrador, encargado de modificar la carta. Por ahora, se ve de la siguiente manera:



En los próximos Sprints iremos modificando las distintas funcionalidades que este tiene.

Al pulsar en el botón de cerrar sesión, la página nos redirige directamente al login de los usuarios, al que se le ha implementado un botón en el header para volver al menú del restaurante y poder visualizar la carta.



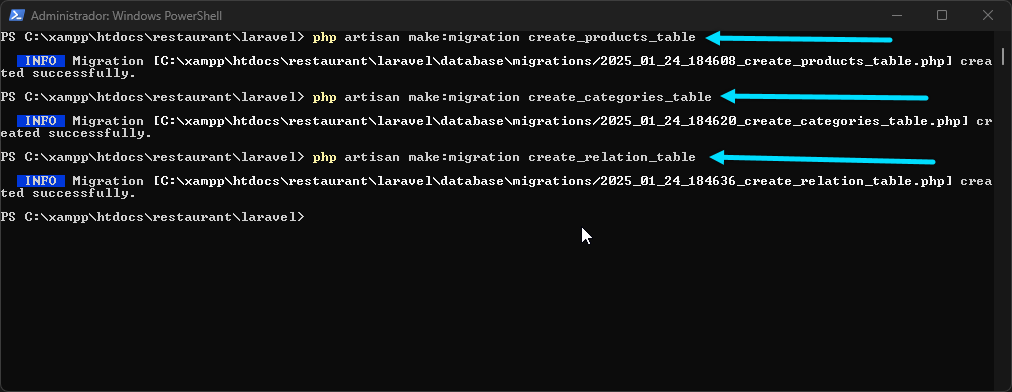
La vista de la carta (vista pública), se ve, por ahora, tan simple como la del administrador, con un botón para acceder al login y entrar a su cuenta. Mas adelante, en los próximos Sprints, se irán introduciendo productos al restaurante y los mostraremos en esta vista.

1. **Modelo y migración inicial:**

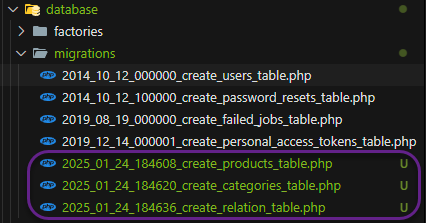
* Crear las tablas principales con migraciones:
* Usuarios (users).
* Productos (productos).
* Categorías (categorias).

Para crear las tablas he abierto la PowerShell de Windows como administrador dentro de la carpeta de laravel, dentro de la carpeta del proyecto (en mi caso llamado “restaurant”), y a su vez dentro de la carpeta de htdocs.

Una vez dentro, debemos escribir los siguientes comandos para las tablas:

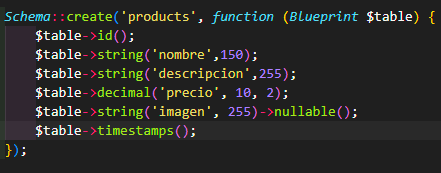


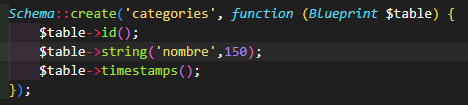
Para comprobar que se nos han creado correctamente, vamos a nuestra carpeta de archivos del proyecto, y, dentro de ella nos vamos a database 🡪 migrations; y comprobamos que están los archivos para las nuevas tablas:

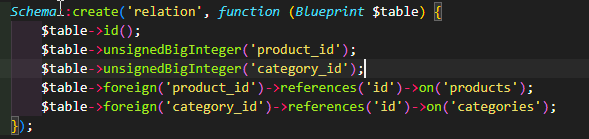


* Implementar relaciones entre las tablas:
* Cada producto pertenece a una categoría.

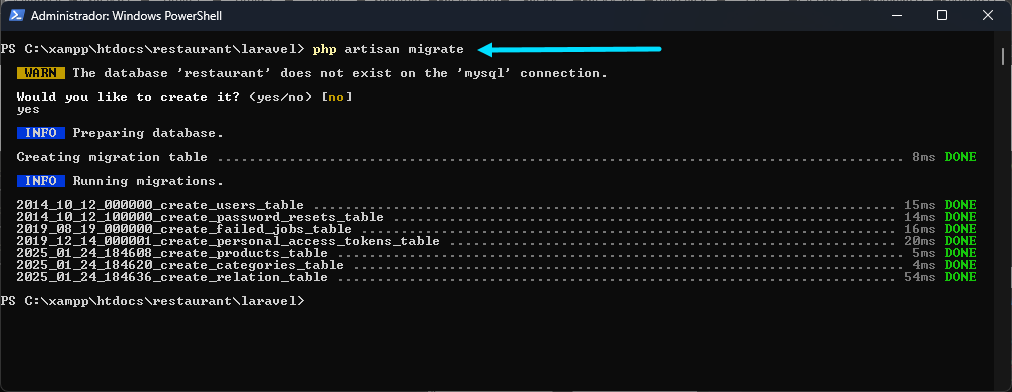
Para crear las migraciones, se han creado los campos que va a tener cada tabla, en mi caso, he elegido los siguientes:



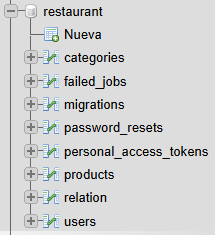




Una vez tenemos esto, debemos irnos a la PowerShell de Windows en modo administrador para ejecutar la migración con el siguiente comando:



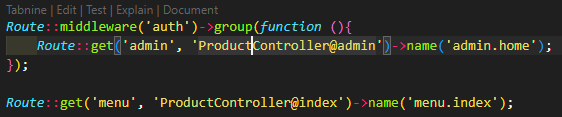
Como podemos ver, nos pide una confirmación ya que no tenemos la base de datos creada, de esta manera, crea la base de datos directamente con todas sus migraciones, y comprobamos que se ha creado correctamente. Nos vamos a phpMyAdmin y vemos nuestra nueva base de datos con la que ya podemos empezar a trabajar:



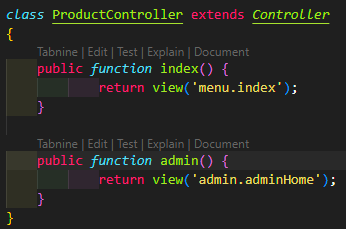
1. **Estructura inicial del proyecto:**

* Organizar controladores, modelos y vistas bajo el patrón MVC.

Por ahora, solo tenemos dos páginas en nuestro proyecto, pero ambas ya se encuentran en un controlador desde donde podemos reorganizar todas nuestras rutas de una manera más clara, en este caso, el archivo web.php se ha quedado de la siguiente manera:

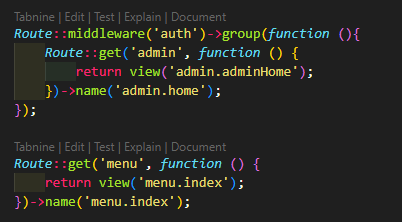


Y el archivo donde controlamos todas las acciones se nos ha quedado de la siguiente manera:



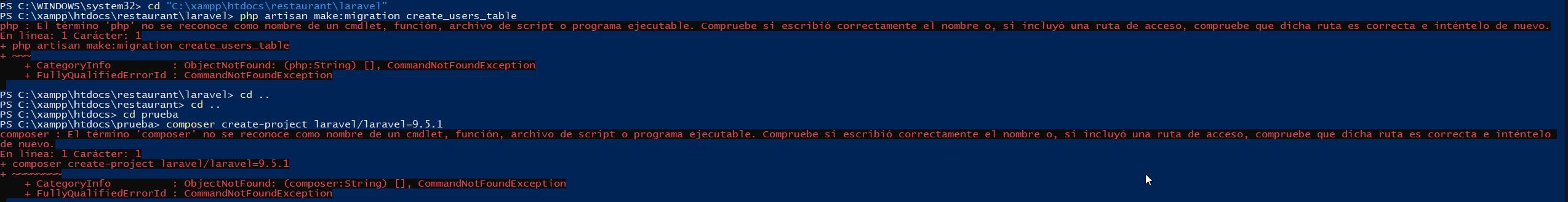
* Implementar middleware para proteger las rutas del administrador.

Como he comentado en el apartado de la seguridad, las rutas de administrador se han protegido con middleware para que ningún usuario ajeno pueda entrar en ellas.

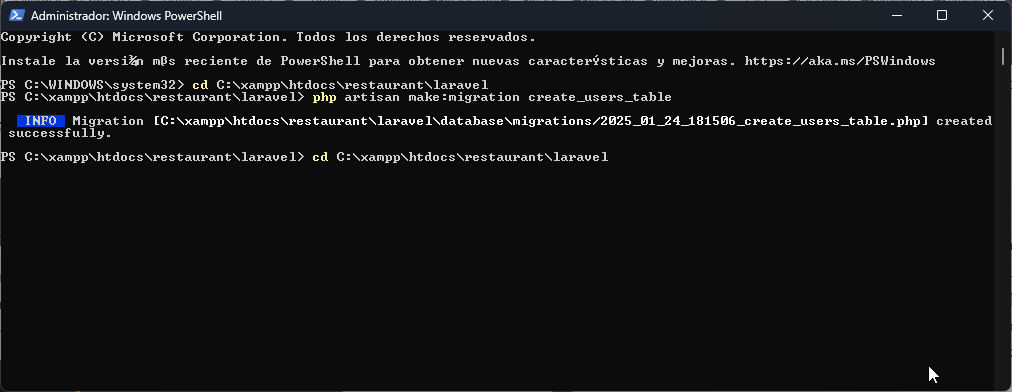


1. **Conflictos**

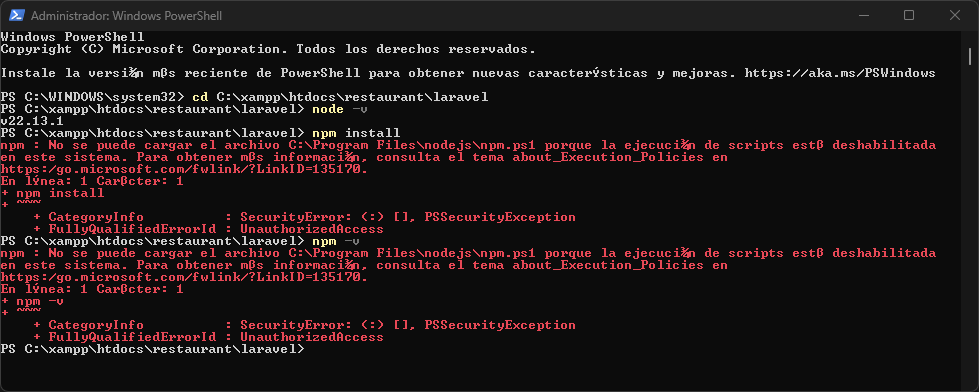
El primer conflicto surgido durante este sprint fue al hacer las primeras migraciones, ya que al hacerlas en casa debíamos tener el “composer” instalado, al no tenerlo, nos surgía el siguiente error:



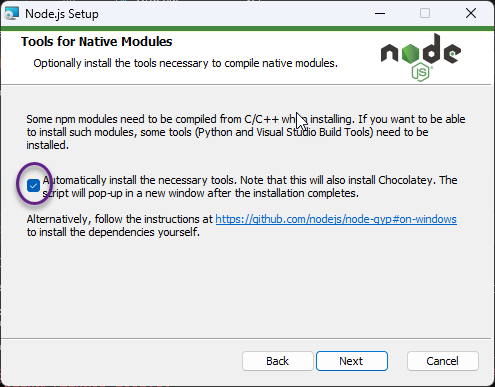
Una vez leído este error, me di cuenta de donde estaba el fallo, ya que en un principio pensaba que ya lo tenia en mi equipo. Una vez instalado el “composer”, volvemos a probar las migraciones de las tablas, y ya nos la hace correctamente como vemos en la siguiente captura:



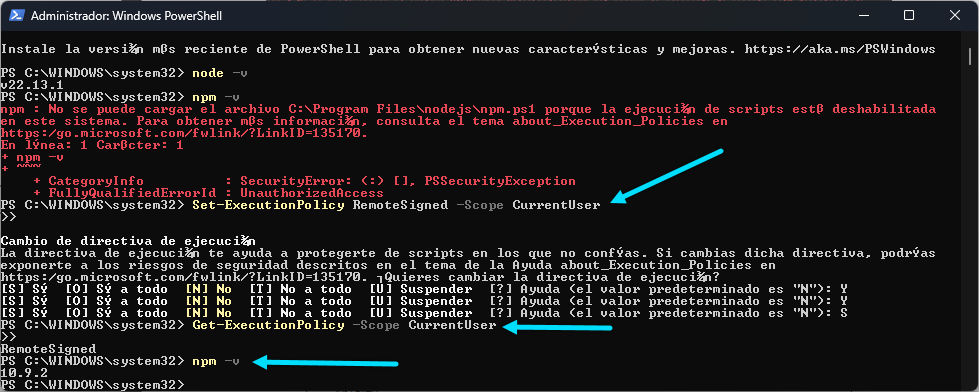
El segundo conflicto surgido fue al ejecutar los comandos necesarios para realizar la instalación de la autenticación. Aparecian los siguientes errores:



Estos errores se deben a que Node.js no está correctamente instalado, por lo que para solucionarlo ejecuté de nuevo el instalador, desinstalé Node.js y lo volví a instalar; sin embargo, esta vez haciendo click en la siguiente casilla, necesaria para que no aparezcan estos errores:

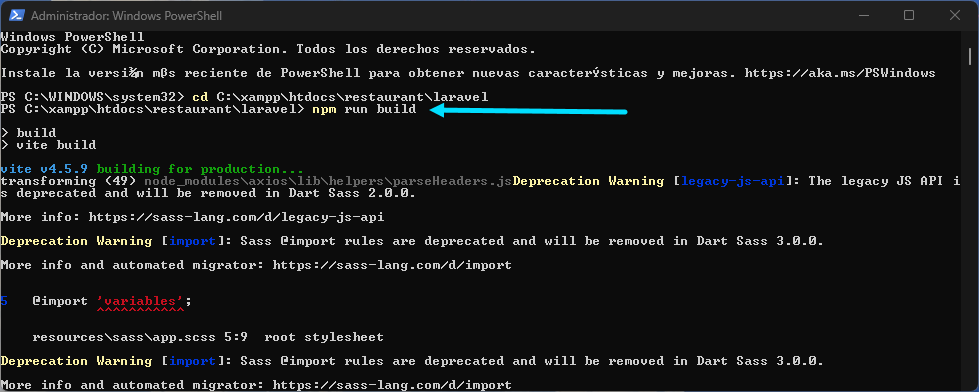


Después de esto, debemos volver a abrir la PowerShell en modo administrador y escribir los siguientes comandos:

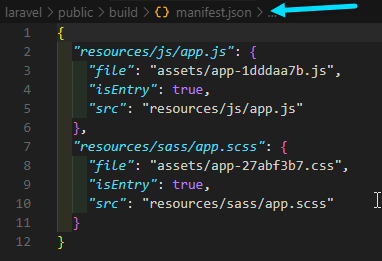


Una vez terminado, podemos comprobar que nos devuelve la versión npm correctamente.

El tercer conflicto surge al intentar entrar en el proceso de autenticación, ya que nos salta un error de Laravel Vite, para solucionarlo, debemos escribir el siguiente comando en la PowerShell de Windows:



Una vez introducido ese comando, recargamos la página y ya nos aparecería nuestro portal de registro, lo que ha hecho este comando es modificarnos algunos archivos y crearnos otros. Un ejemplo sería el siguiente, el cual lo podemos ver en la ruta señalada.



Vista de la carta: <http://localhost/restaurant/laravel/public/menu>

Vista del administrador: <http://localhost/restaurant/laravel/public/login>

# Sprint 2: Gestión de categorías y productos

**Objetivo**: Implementar el sistema CRUD para productos y categorías, asegurando validaciones y control de datos.

**Tareas**:

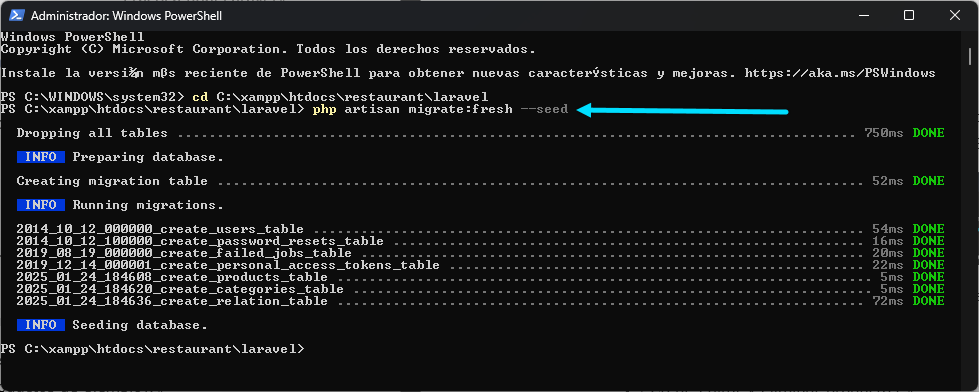
1. **Migraciones y seeders:**

* Crear migraciones para categorias y productos (si no están completas en el Sprint 1).

Estas migraciones ya están explicadas en el Sprint 1, en el punto 3.

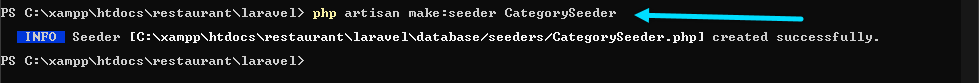
* Crear seeders para poblar la base de datos con datos iniciales (ej.: categorías predefinidas y algunos productos de ejemplo).

Antes de crear los seeders, podemos reiniciar la base de datos para probarla, por si tenemos datos de antes de pruebas que he ido haciendo. Para ello, utilizaremos la siguiente línea de comandos en la PowerShell de Windows:

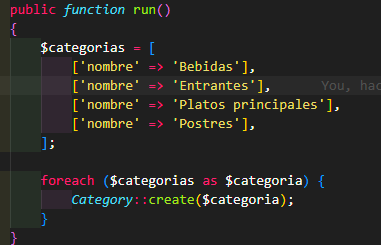


Una vez reiniciado la base de datos, también debemos crear un nuevo usuario utilizando el sistema de registro que hemos implementado previamente.

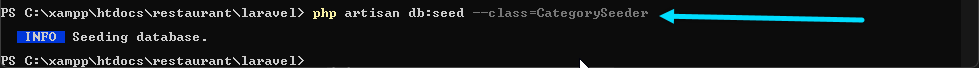
Para hacer los seeders, debemos ejecutar en la PowerShell un comando para crear los archivos que llevaran datos por defecto, para después, en esos archivos, crear los datos. Vamos a ver un ejemplo para crear las categorías. Primero creamos el archivo con la consola de comandos:



Después, ponemos los datos que queremos insertar en nuestra base de datos en el archivo:



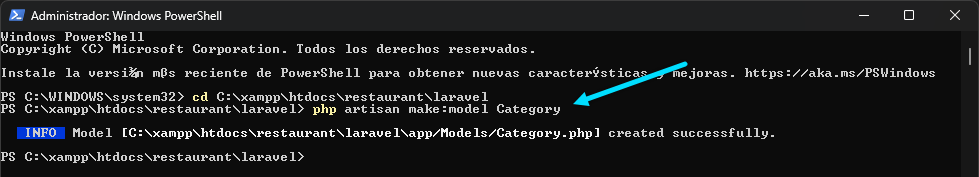
Una vez tenemos los datos que queremos implementar, ejecutamos la siguiente línea de comandos en la PowerShell, de manera que se transfieran todos los datos a la base de datos.



Esto es un ejemplo para poblar una tabla concreta. Para poblar toda la base de golpe usaremos un comando diferente (ver punto 5 del apartado “Como desplegar la base de datos”). De la misma forma, hacemos lo mismo para la tabla de productos.

1. **CRUD de categorías:**

Para seguir con las siguientes tareas, debemos crear un modelo y el controlador para las categorías con las siguientes líneas de comandos en al PowerShell de Windows:



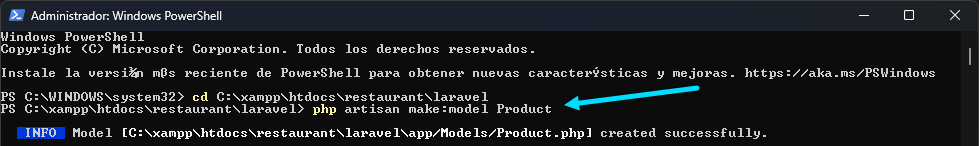


Una vez tenemos esto hecho, se nos habrá creado en nuestro proyecto unos archivos con los nombres que hemos indicado. En el controlador para las categorías es donde se deben colocar los métodos para: ver el listado de categorías (index); ver la página con el formulario para crear una nueva categoría (create); guardar la nueva categoría (store); ver la página con el formulario para editar una categoría (edit); guardar la modificación para la categoría editada (update); eliminar una categoría (destroy). A todas estas opciones para las categorías accedemos a través de un botón en la vista de administrador.

También he implementado aquí el método para mostrar los productos en la vista pública al hacer uso del modelo “Category”, (vistaPubli).

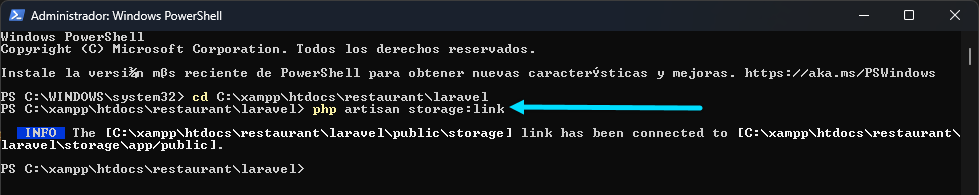
1. **CRUD de productos:**

Al igual que con las categorías, para seguir con las siguientes tareas, debemos crear un modelo para los productos con la siguiente línea de comandos en al PowerShell de Windows:



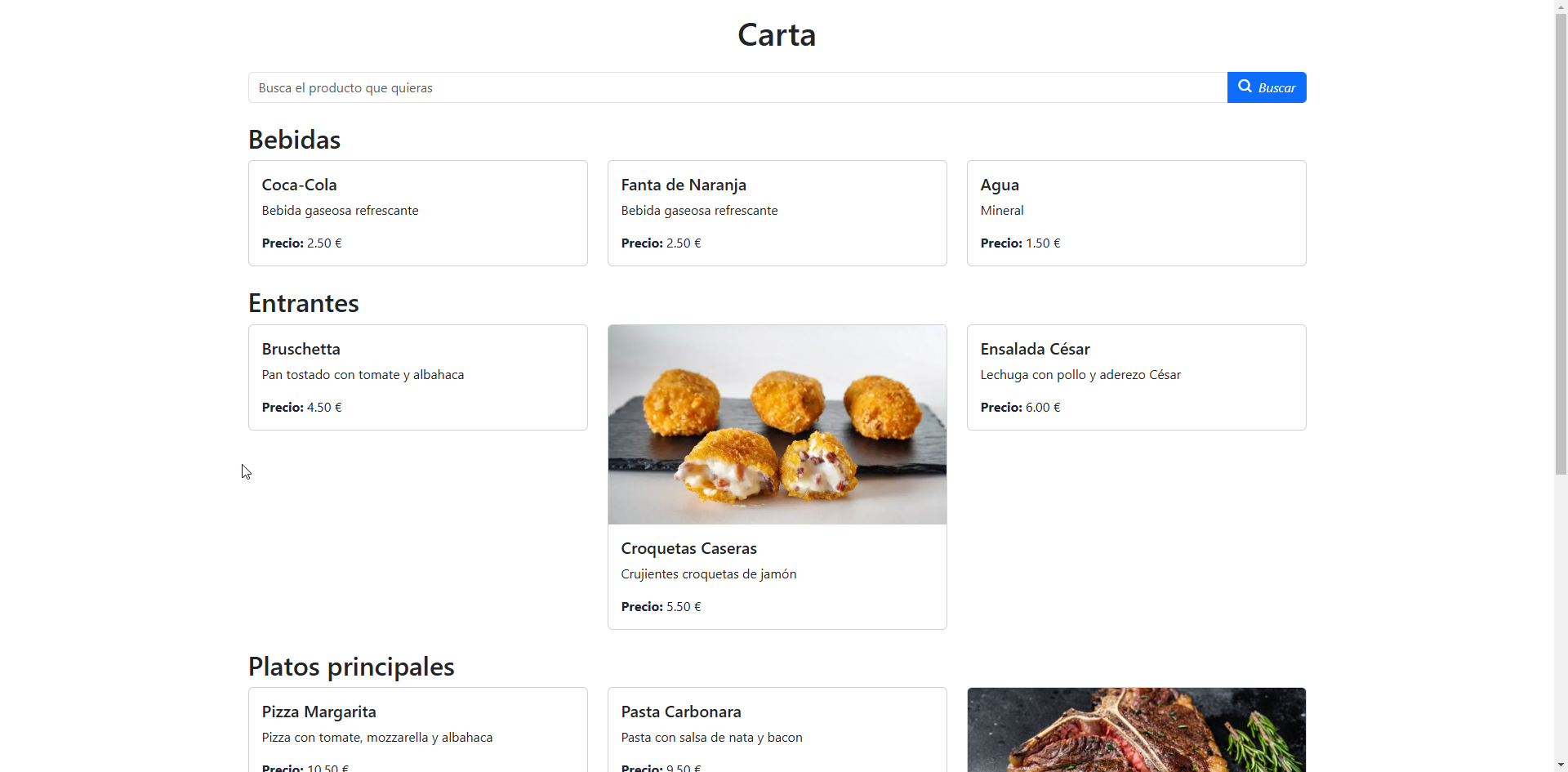
Después de ello, se nos creará un archivo para este modelo en la ruta app 🡪 Models. Al igual que en las categorías, debemos crear también el controlador, y, en él, colocaremos los métodos para: ver la vista de inicio del administrador (admin); ver el listado de los productos organizados en categorías (verProductos); ver el formulario para crear un nuevo producto (create); guardar el nuevo producto (store); ver el formulario para editar un producto (edit); guardar la modificación para el producto editado (update); eliminar un producto (destroy).

Para subir las imágenes, utilizamos en primer lugar un input de tipo archivo. Esta imagen se guardará en la ruta storage 🡪 app 🡪 public 🡪 productos (esta última carpeta creada por mí). A la hora de listar estas imágenes, debemos escribir un comando en la PowerShell para que las imágenes de esta carpeta se guarden en la carpeta public, a la cual accedemos para ello. Dicho comando es el siguiente:



1. **Integración con vistas públicas:**

Actualmente, en la vista pública tenemos la siguiente organización por categorías, comenzando por las bebidas.



Como podemos ver, en la parte superior tenemos un buscador conectado con el método de listar los productos en esta vista en el controlador de categorías, en el cual podemos filtrar introduciendo el nombre de un producto cualquiera, y nos devolverá los productos que contengan la cadena de caracteres que hemos introducido.

1. **Conflictos:**

En este sprint, al trabajar todo el tema de migraciones de tablas, vistas, seeders, creaciones y ediciones, etc., se ha trabajado mucho con variables con nombres parecidos, como, por ejemplo, el uso de: productos – producto – products. Al haber estas variaciones en las palabras, en algunas ocasiones no estaban bien las redirecciones a las rutas, métodos u otros. Sin embargo, estos problemas se solucionan con un poco de paciencia, siguiendo el camino que te dice el propio Laravel (variable no encontrada, por ejemplo), hasta dar con la línea que había que corregir.

Vista de la carta: <http://localhost/restaurant/laravel/public/menu>

Vista del administrador: <http://localhost/restaurant/laravel/public/login>

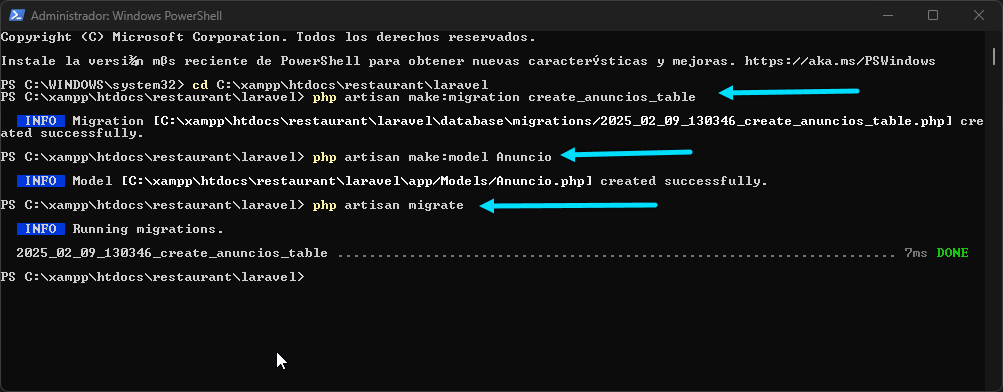
# Sprint 3: Gestión de anuncios y sistema de notificaciones

**Objetivo**: Permitir a los administradores que al publicar noticias se manden mensajes automáticos a través de Telegram.

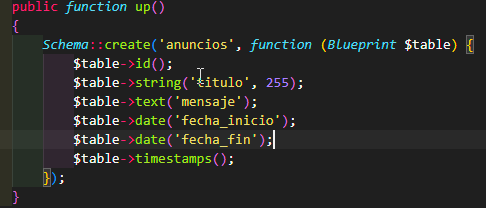
**Tareas**:

1. **Modelo y migración de anuncios:**

Para crear la tabla, antes hemos creado la migración y el modelo para los anuncios en la PowerShell de Windows:



Antes de ejecutar la migración del último comando de la foto de arriba, ponemos cuales son los campos a tratar en esta tabla, de la siguiente manera:



Una vez hecho esto, ya podemos ejecutar la migración.

1. **CRUD de anuncios:**

Para el CRUD de anuncios se han seguido los mismos pasos que en las categorías y en los productos. Se ha creado el controlador; en el archivo de rutas se han definido todas las necesarias para apuntar al controlador; y, en este, se han definido las distintas acciones que debemos hacer con los anuncios (listar, crear, editar y eliminar). A la misma vez, a todas estas acciones pueden a través de un card que se he ha creado en la vista del administrador, para que sólo él tengo acceso a estas modificaciones.

1. **Vista pública de anuncios:**

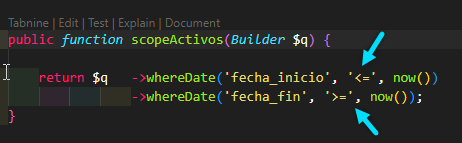
* Mostrar anuncios activos en la parte superior de la página pública.

Una vez creados los Anuncios de prueba gracias a lo hecho en el punto anterior, comprobamos en nuestra vista pública que los anuncios se nos ven correctamente:

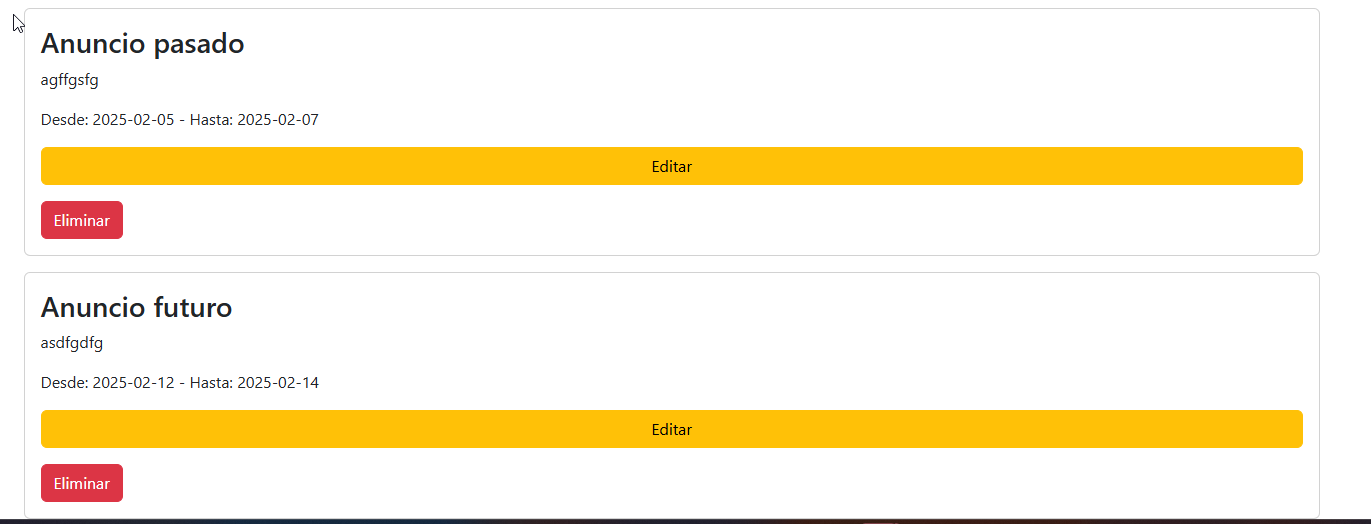


* Implementar una consulta para filtrar anuncios según la fecha actual.

Para asegurarnos que los anuncios que se muestran son solo los que están activos y válidos, se ha hecho la siguiente consulta, en la que nos aseguramos que la fecha de inicio del anuncio es menor o igual la fecha actual, y la fecha de fin del anuncio es mayor o igual a la fecha actual.



Como podemos comprobar, he creado dos anuncios (uno pasado y otro para el futuro), y ,en la vista pública, estos dos anuncios no nos aparecían.



1. **Notificaciones por Telegram:**

* Configurar un bot de Telegram.

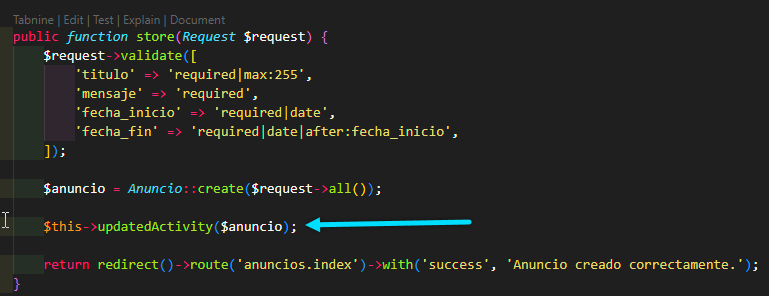
Para la configuración del bot de Telegram se han seguido las instrucciones vistas en clase, siguiendo los pasos para llegar al punto final y comprobar que los mensajes se envían correctamente.



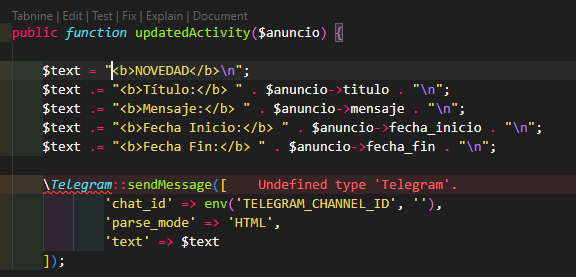
En mi caso, el canal y bot creados los he llamado como el nombre del restaurante: Mesón Jiménez (que modificaré más adelante en las demás páginas).

* Implementar funcionalidad para enviar mensajes automáticamente al publicar un anuncio.

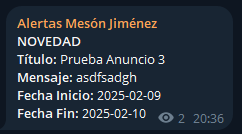
Para implementar esta funcionalidad, nos hemos ido al controlador de los anuncios, y, en la función para guardar un nuevo anuncio en la base de datos, se ha mandado este a la función que usaremos para enviar el mensaje.



Seguidamente, esta función la hemos modificado de la siguiente manera desde el ejemplo visto en clase, para que los mensajes y anuncios salgan de la manera que queremos (más adelante cambiaremos la estética de los mismos).



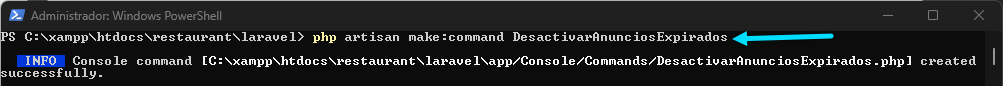
Como podemos ver, los mensajes nos saldrían de la siguiente manera en Telegram:



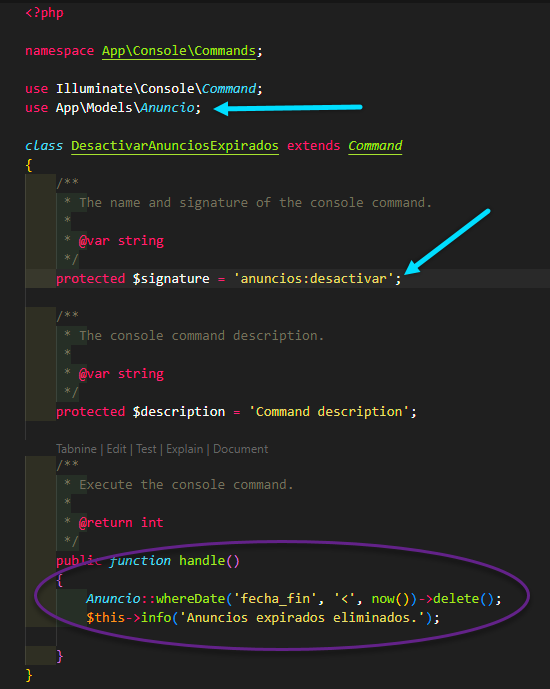
1. **Laravel Scheduler o similar:**

* Configurar el scheduler o similar para desactivar automáticamente los anuncios expirados.

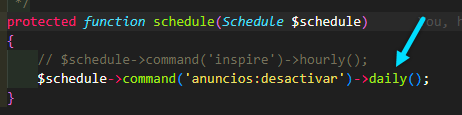
Para configurar Laravel Scheduler debemos en primer lugar ejecutar el siguiente comando en la PowerShell:



Una vez ejecutado, se nos creará el siguiente archivo en la ruta app 🡪 Console 🡪 Commands 🡪 DesactivarAnunciosExpirados.php; este archivo debemos configurarlo de la siguiente manera:



Además de este archivo, también debemos modificar el archivo Kernel.php, en la misma carpeta de Commands. En él nos aparecerá la función Schedule, a la que debemos introducirle el siguiente comando para que se ejecute diariamente:



Ahora que tenemos el comando configurado, debemos iniciarlo para que se ejecute, para ello usamos el siguiente comando:



* Asegurarse de que las tareas programadas se ejecuten correctamente.

Para asegurarnos que el comando programado se ejecuta correctamente sin necesidad de esperar a que expire un anuncio, podemos ejecutar el siguiente comando:



Si nos aparece ese mensaje, que es el configurado en la función handle, es porque el comando funciona correctamente.

1. **Conflictos:**

El único problema destacable que me ha surgido en este Sprint ha sido que, al ejecutar por primera vez este comando:



No me lo detectaba. Después de investigar, me he dado cuenta que fue porque en el archivo de DesactivarAnunciosExpirados.php, no había modificado el nombre de la variable $signature, por lo que no me lo reconocía. En un primer momento esta aparece inicializada como “command:name”.

Vista de la carta: <http://localhost/restaurant/laravel/public/menu>

Vista del administrador: <http://localhost/restaurant/laravel/public/login>