Yolanda Morales Zamora
15/05/2023
DWES
IES Punta del Verde. Sevilla

Índice

1. **Resumen información proyecto.**
   Nombre del proyecto: Knowbuysell.
   Framework utilizado: Laravel, el framework de los artesanos de la web.
   Lenguaje de programación: PHP

2. **Información básica Laravel & PHP.**
   Laravel es un framework de desarrollo web de código abierto escrito en PHP. Fue creado por Taylor Otwell y se lanzó por primera vez en 2011.
   Basado en el patrón de desarrollo MVC (modelo-vista-controlador), que implica la modularización de los componentes y partes del software. Esto te permite separar elementos de la vista y del diseño, de las estructuras de datos, y del código orientado a la lógica. Separa las aplicaciones en pares modulares, más pequeñas, que pueden modificarse sin alterar los componentes.
   PHP es el lenguaje del lado del servidor, es decir, no corre directamente como lo hace HTML, CSS o Javascript, sino que necesita un servidor web para ser ejecutado, como puede ser Apache, entre sus características es un lenguaje con facilidad para crear un servidor y levantar el sitio.
   Investigando hemos descubierto que existen distintos paquetes que permiten crear un servidor de desarrollo con varias herramientas que serán útiles, como por ejemplo: EasyPHP, paquete completo que contiene en su instalación diversas herramientas para el desarrollador, entre ellas, el lenguaje PHP, MySQL, el motor de bases de datos no relacional MongoDB, los servidores Apache y Ngimx, Xdebug, el gestor de bases de datos PostgreSQL y WordPress, entre otras tecnologías interesantes.
   Xampp permite instalar rápidamente PHP, MySQL, Apache y TomCat (servidor para lenguaje Java en la web).
   ORM (Object Relational Mapping) o mapeo de objetos realacional, es la técnica por medio de la cual se pueden abstraer las consultas SQL o métodos de clasese.
   Los modelos en el MVC son clases que permiten moldear cómo serán los objetos de las clases de bases de datos y cómo puede ejecutarse operaciones sobre ellos.
   El ORM de Laravel es ELOQUENT, componente que permite abstraerse de las consultas SQL tradicionales y trabajar con bases de datos de manera mucho más simple.
   También existe lo que se conoce como Scaffolding, técnica provista por varios entornos de trabajo modernos, que se basan en generar componentes de software de manera automática para facilitar tareas al desarrollador, creando controladores, vistas, modelos y migraciones, entre muchas otras cosas.

3. **Recorrido simplificado de instalaciones.**
   <u>Xampp / Composer:</u>
   En este informe no vamos a repetir cómo se tiene que instalar Xampp o Composer nos remitimos la documento “TareaLaravel.pdf”.
   Partimos desde el punto en el que Xampp y Composer ya los tenemos instalados. Al igual que Visual Studio Code.

Al igual que Python o Javascript tiene paquetes como pip o npm, en Composer encontramos un gestor de paquetes y dependencias por excelencia dePHP.
Cuando Xampp y Composer ya se han instalado, nos situamos en el terminal de de VSC (ctrl+ñ) y utilizamos los siguientes comandos:

php-v
composer –version

Nos servirán para saber qué versión de php tenemos instalada al igual qué versión de Composer.

Crear proyecto Laravel:
Nos vamos a la carpeta 'htdocs' dentro de la ruta de Xampp, y allí abrimos la consola y en ella ejecutamos:

composer create-project laravel/laravel mi-primer-app

Este comando se encarga de visitar el repositorio oficial de laravel en GitHub, para buscar la última versión estable del proyecto, y clona en tu pc o portátil una versión limpia en una carpeta con el nombre que coloques después de la instrucción "laravel/laravel".
Instala también todas las dependencias necesarias para el framework.
El navegador es: localhost/mi-primer-app/public con el servidor Apache encendido en el panel de control (Xampp). Aparecerá la pantalla de laravel de su vista "welcome.blade.php".
En el terminal si no estamos en la carpeta del proyecto, usamos el comando cd mi-primer-app y después php artisan serve.
El localhost es 8000
Además del VSC existen otro entorno de desarrollo integrado para PHP llamado PHPStorm que es de pago, excepto para las universidades.

Bootstrap:

Ejecutamos el siguiente comando en nuestro VSC:
composer require laravel/ui
php artisan ui bootstrap
npm install
npm run dev

Esto nos va a permitir llamar a las librerías de forma dinámica.
Npm es el gestor de paquetes Node.js, un software destinado a ejecutar código JS por fuera del navegador, dentro de la computadora e incluso, a crear servidores basados en esta tecnología.

Tenemos que recordar que el paquete php artisan y los paquetes Composer tienen como beneficio que limitan la cantidad de peticiones HTTP realizadas mediante el uso de un CDN, con o cual la aplicación funcionará con mucha más rapidez y utilizará menos recursos si precompila código de esta manera.

4. **¿Cómo incluimos Bootstrap en Laravel una vez instalado el paquete correspondiente?**

La forma más sencilla de incluir las librerías de estilos es por medio de un CDN (Content Delivery Network).
En el enlace https://getbootstrap.com/docs/5.0/getting-started/introduction/ justo debajo de donde pone Quick start, copiamos el código que nos aparece y se coloca dentro de la etiqueta <head></head>
Bootstrap está basado en el sistema de columnas resposive o adaptables, que permiten trabajar con sitios mobile first, es decir, que se adaptan fácilmente a cualquier tipo de pantalla, poniendo el foco en los dispositivos que por estadística más navegan la Web: los smartphones y los equipos móviles. Se adaptan las filas y las columnas porque están basadas en flexbox.

Ancho pantallas, que hemos utilizado para que la aplicación sea Responsive, de esta forma si abrimos la aplicación desde el móvil se verá acorde con su tamaño y no se desvirtuará:

1. Extra-Small: "col" smarphones o teléfonos inteligentes < 576
2. Small: "sm" entre laptop y teléfono [576 – 768]
3. Medium: pantallas amplias o laptop pequeñas: "md" [768-992]
4. Large: pantalla superior a 992 píxeles pero menos de 1200 "lg"
5. Extra-Large: pantalla superior 1200 "xl"

Los contenedores de Bootstrap permiten englobar contenido con un margen predefinido. Cada sección debe estar en su propia fila. El grid de Bootstrap divide el ancho de la pantalla en 12 columnas, puedes indicara cada elemento que tome la cantidad de columnas que desees.

Ejemplos: total 12 col

Mitad 6 col

12 ó 9 para pequeño

Librerías anteriores a Bootstrap (basado en CSS) son JQuery, Bundle y Popper.

Hemos tenido en cuenta también las clases para definir el relleno de un elemento :

p: con unidades rem (realtivas)
m: margen
pb: padding-bottom
mb: margin-botton

Como Bootstrap está basado en flexbox (CSS3) consigue ser Responsive.
Flex-wrap: define que los elementos anidados en sus interior pueden apilarse uno sobre otro en caso de que la pantalla lo requiera por su ancho.
Justify-content-between: indica que los elementos deben ir en el orden en que son declarados y tienen que estar distribuidos uniformemente en el ancho.

5. **Hacemos uso de la herencia de php para ficheros Blade.php.**
En este apartado debemos tener en cuenta las vistas en laravel se encuentra en la carpeta "views", con las plantillas .blade realizamos los modelos de cada página. Desde le fichero Web.php es la carpeta "routes", desde donde vamos a devolver con una función anónima ("/compras") para que en la vista en nuestro caso "compra.index", "compra.create", "compra.edit" y "compra.form", gestionemos las vistas de la aplicación

en este caso en concreto las mencionadas son para la gestión del CRUD, tenemos también las vistas "welcome" para la vista de inicio.

Todo esto se debe a que si la aplicación crece, no podemos llamar a la librería en cada archivo nuevo, rompería la filosofía de Laravel, para subsanarlo podemos usar el sistema de herencia que ofrece el motor de plantillas Blade, con herencia en sus vistas.

- Primero, tenemos que definir la plantilla como vista base o padre, de la que van a heredar el resto. Colocamos la plantilla base en la carpeta "views" dentro de la subcarpeta "layouts" donde se colocan los templates que se usan como base. La ruta completa del archivo será c:Xampp/htdocs/resources/views/layouts.
- En segundo lugar la plantilla en la que se va a recibir la herencia debemos establecer **@extends()** dentro del paréntesis meteríamos la ruta del fichero padre, indicamos la vista que debe heredar de otra, y toma como parámetro a ruta para encontrar el archivo base o padre, como hemos indicado.

6. El camino realizado con Laravel.

- .env : donde tenemos que comprobar que los datos de la base de datos son correctos para poder realizar la conexión.
- config.session.php : donde tenemos que configurar previamente si es necesario, para las cookies.
- views.welcome.blade.php : fichero de entrada a la aplicación en el cuál lo primero que va a aparecer es la aceptación de las cookies.
- views.usuarios.blade.php: fichero para mostrar los usuarios de la aplicación, contiene la herencia @extends(layouts.base) y un foreach para poder sacar la información de cada usuario y a través de las etiquetas html serán mostradas.
- views.layouts.base.blade.php: en este fichero insertamos el CDN de Bootstrap y damos formatos al resto de vistas, incluimos @yield(content) que nos va a servir para que se pueda insertar contenido particular en cada vista.
- views.layouts.auth-master.blade.php: damos formato para las vistas del formulario relacionadas con el login y register.
- views.layouts.partials.navbar.blade: damos formato a la barra de navegación de la aplicación.
- app.Http.Controllers: es la carpeta donde guardaremos cada fichero controlador que hemos creado para que de la instrucción necesaria.
- app.Http.Models: es la carpeta del proyecto donde se van a guardar las clases, que lo habitual es que coincida con cada tabla de la base de datos.
- config.app.php: fichero en el que hemos realizado modificaciones para configurar correctamente los datos necesarios para la gestión de generar pdf en la aplicación.
- config.session y config.auth: ficheros donde se configuran las modificaciones necesarias para la autenticación del usuario.
- database.migrations: carpeta donde se guardan todas las migraciones que gestiones a la base de datos.
- resoruces.views: carpeta donde se guarda o estructuran todas las vistas que necesitasmos generar para la aplicación.
- routes.web.php: fichero muy importante, ya que si no se establecen correctamente las rutas, no podremos acceder a las vistas, ni a los métodos, ni al resto de recursos de Laravel.
- public.assets: carpeta donde incluimos la referencia de Bootstrap en la carpeta CSS y la carpeta JS.

- vendor: carpeta donde podemos comprobar todos los paquetes hemos instalado, por lo tanto, podremos ver si nos faltan paquetes que podamos utilizar.

7. Creación de clases.

Para crear las clases tenemos "artisan" creamos modelos a través de php artisan make:model Departamento -m. Se encarga de crear la clase con el nombre que le demos y -m creará una migración correspondiente al modelo.
La clase que acabamos de crear y el método up() que se encuentra dentro del código son los encargados, tras ejecutar un determinado comando, de crear una tabla en la base de datos que hemos creado.
El método tiene en su interior una llamada al método estático create, de la clase Scheema, que se encarga de generar una nueva tabla por medio de una función anónima, se le pasa los campos que dicha base de datos debe tener. En este caso, un campo id, que se define por el método id() y especifica que dicho campo es de tipo AUTO_INCREMENT, NOT NULL y PRIMARY KEY de la tabla; por otro lado, timestamp(), las etiquetas de tiempo, dos columnas que almacenan la fecha en la cual fue creada y la fecha en la cual se modificó el registro por última vez.
Creamos los modelos de ventas y de compras
Existen dos formas de crear las clases poniendo al final -m y sólo creará la clase ó poniendo al final -mcr y creará la clase, el controlador y otros recursos.

EJEMPLO: php artisan make:model Venta -mcr

Crea el modelo, el controler y recursos

En los ficheros controller creamos los métodos para nuetro CRUD

Las vistas se añaden en la carepta views y las rutas en la routes.Web

EJEMPLOS:

a. php artisan make:model Departamento -m
b. php artisan make:model Venta -m
c. php artisan make:model Compra -m

Aunque creamos las clases en singular con letra mayúscula, Laravel crea las migraciones en plural y minúsculas, respetando las convenciones para que todo el mundo pueda entender los códigos.
User: (Models): con el nombre departamento_id, cada usuario utilizará como clave foránea este dato, que Eloquent usará para buscar.

8. Creación de migraciones.

Si necesitas modificar una tabla existente en Laravel después de haber ejecutado las migraciones, puedes utilizar las migraciones de Laravel para realizar cambios en la estructura de la tabla de manera controlada.
A continuación, te explico los pasos a seguir para modificar una tabla existente mediante migraciones en Laravel:
PASAMOS a realizar la migración, php artisan migrate
MUY IMPORTANTE TENER LA BASE DE DATOS CREADA ANTES DE EJECUTAR ESTE COMANDO PARA QUE NO NOS APAREZCA EL ERROR: Unknow database.

Crea una nueva migración: Para modificar una tabla existente, debes crear una nueva migración utilizando el comando php artisan make:migration. Por ejemplo, para agregar un nuevo campo a una tabla, puedes ejecutar el siguiente comando:
bash
```
php artisan make:migration add_new_field_to_table --table=nombre_tabla
```
Esto creará un nuevo archivo de migración en el directorio database/migrations con un nombre único y descriptivo.

- Edita la migración: Abre el archivo de migración creado en el paso anterior y utiliza los métodos up() y down() para definir los cambios que deseas realizar en la tabla. Por ejemplo, para agregar un campo a la tabla, puedes utilizar el método addColumn():
php
```
public function up()
{
   Schema::table('nombre_tabla', function (Blueprint $table) {
      $table->string('nuevo_campo');
   });
}

public function down()
{
   Schema::table('nombre_tabla', function (Blueprint $table) {
      $table->dropColumn('nuevo_campo');
   });
}
```
En el método up(), utilizamos el método addColumn() para agregar un nuevo campo a la tabla. En el método down(), utilizamos el método dropColumn() para revertir el cambio en caso de que sea necesario realizar una migración de rollback.
- Ejecuta la migración: Una vez que hayas editado la migración, puedes ejecutar el comando php artisan migrate para aplicar los cambios en la tabla:
bash
```
php artisan migrate
```
Esto ejecutará todas las migraciones pendientes, incluyendo la nueva migración que has creado.
Laravel realizará los cambios en la tabla de acuerdo con la migración que has creado. Si necesitas realizar más modificaciones en la misma tabla, puedes crear migraciones adicionales y ejecutar el comando php artisan migrate para aplicar los cambios.
Recuerda que las migraciones de Laravel te permiten mantener un registro de los cambios realizados en la estructura de la base de datos y realizar actualizaciones de manera controlada. También puedes utilizar los comandos php artisan migrate:rollback y php artisan migrate:refresh para revertir o refrescar las migraciones, respectivamente.

Deshacer migración:
Una migración rollback (reversión de migración) en Laravel se refiere a deshacer los cambios realizados por una migración previamente ejecutada. Es una forma de revertir la última migración o un conjunto de migraciones a un estado anterior en la base de datos.

Cuando ejecutas una migración en Laravel, se realiza una modificación en la estructura de la base de datos, como crear una tabla, agregar un campo, modificar un índice, entre otros cambios. Si necesitas deshacer esos cambios, puedes utilizar el rollback.

9. Creación de Controllers.

Código ejemplo para crear un fichero controlador:

Php artisan make:controller Compras -resource.

COOKIES:

Creamos un controller para nuestra página de Bienvenida

php artisan make:controller WelcomeController --resource

10. ¿Cómo puedo ver qué paquetes están instalados?

En el proyecto tenemos una carpeta llamada vendor que contiene todas las dependencias instaladas a través de Composer. Dentro de esta carpeta, se encuentra la carpeta correspondiente al paquete que deseamos verificar. Si encuentramos la carpeta del paquete, significa que está instalado.

11. 'Session' en Laravel.

En Laravel, cuando se utiliza el sistema de autenticación `Auth` incorporado, se utiliza un mecanismo llamado "tokens de autenticación" para mantener la sesión del usuario. Estos tokens son almacenados en cookies o en el almacenamiento de sesión de Laravel.

Cuando un usuario inicia sesión en Laravel, se crea un token de autenticación único para ese usuario y se almacena en la sesión. Esta sesión se asocia con la cookie del navegador del usuario. Cuando el usuario realiza solicitudes posteriores al servidor, el token de autenticación se utiliza para verificar si el usuario está autenticado.

El comportamiento por defecto de Laravel es que un token de autenticación solo es válido para una única sesión o navegador. Esto significa que, si el usuario inicia sesión en un navegador y luego intenta abrir una sesión en otro navegador, el segundo inicio de sesión invalidará automáticamente el token de autenticación anterior, cerrando la sesión en el primer navegador.

Este comportamiento es controlado por Laravel a través del middleware `StartSession` y la configuración de sesión en `config/session.php`. La configuración `driver` en el archivo `session.php` determina dónde se almacenan las sesiones (por ejemplo, en archivos, en la base de datos, en Redis, etc.). Además, la opción `same_site` puede configurarse para

determinar si la cookie de sesión se puede compartir entre diferentes sitios o si se limita al mismo sitio.

Si queremos personalizar este comportamiento, podemos modificar el archivo `session.php` y ajustar la configuración según nuestras necesidades. Por ejemplo, podríamos permitir múltiples sesiones simultáneas configurando la opción `expire_on_close` a `false` y utilizando un driver de sesión compatible con múltiples sesiones, como la base de datos o Redis.

Es importante tener en cuenta que esta es la configuración predeterminada de Laravel, pero es posible modificarla o reemplazarla para adaptarse a tus requerimientos específicos de autenticación y sesiones.

12. Comprobar rutas activas.

Podemos saber qué rutas tenemos activas a través del comando:

```
Php artisan route:list
```

Con este comando vemos los métodos y si se acede por POST etc… Para acción con la base de datos te indica qué método debemos usar.

13. Reporte de errores solventados.

a. Importante: en las rutas tiene que estar bien identificado cada fichero de vistas.
b. También es importante que se hagan las importaciones en el fichero de controlador de los paquetes o clases que tengamos que usar para que no nos dé fallo la aplicación.
c. Instalé el paquete Breeze, pero lo tuve que desinstalar porque sospecho que no era compatible con otros paquetes que ya tenía instalados.
d. Tuve que eliminar tablas a través de Laravel para poder corregir un error que me apareció.
e. En el siguiente código también tuve error en la aplicación sólo porque se me había pasado incluir la palabra 'nombre' dentro de los paréntesis de string(), por lo que no me dejaba migrar la información:

```
public function up(): void
{
Schema::créate('departamentos', function(Blueprint $table){
$table->id();
$table->string('nombre');
$table->timestamps();
});
}
```

f. Importante tener claro los campos de la base de datos para no tener que modificar tanto con las migraciones(descripcionproducto).
g. Otros errores ya solventados.

14. Objetivos alcanzados.

Hemos incluido cookies.
Nuestro Login está correcto.
El usuario mete su clave y ésta queda encriptada en la base de datos.
El apartado de crear cuenta.
La sesión no se puede abrir en otro navegador pero si es el mismo se mantiene abierta.
La conexión con la base de datos es correcta.
Hemos creado usuarios y se han almacenado en la base de datos de forma correcta.
Hemos realizado un CRUD en cuanto a las compras realizadas por los usuarios, cada usuario que entra puede crear una compra, modificar la compra y eliminar la compra.
Hemos conseguido generar un documento pdf del listado de compras.
Hemos generado nuestros métodos que necesitábamos dentro del proyecto en PHP.

15. Gestiones pendientes.

Me gustaría poder completar las clases y pulir cosillas que tengo que modificar, como por ejemplo que a cada usuario le aparezca su lista de la compra y reorganizar mejor la aplicación.
Enlazar las tablas porque no están relacionadas.
Completar y mejorar las cookies.
Realizar la Grafica con Laravael.
Incluir imágenes con Laravel en la aplicación y que se guarden en la base de datos.

16. Dudas

DUDAS: ¿para qué sirve el token que se genera con el csrf?
Por qué sale NULL en la base de datos con el timestamp??

17. Conclusiones.

He aprendido muchas cosas en muy poco tiempo, si no hubiera estado trabajando por las mañanas, entre otras cosas, creo que podría haber avanzado más el proyecto. Espero y deseo hacerlo.
Aunque al principio de empezar el proyecto con Laravel tenía la sensación de estar escalando una montaña, una vez que conseguí entender el mvc, y entender un poco más el funcionamiento de Laravel, reconozco que me gusta, a pesar de que siempre me haya gustado más Java o Python, reconozco que al final cuando lo entiendes, lo trabajar y más o menos lo vas controlando, es cuando empieza a gustar.

Reconozco que no fue nada fácil para mí al principio, peor a base de poner empeño, he podido obtener algunos frutos al respecto.

Igualmente soy muy consciente de que aún me queda mucho por aprender y me encantaría hacerlo.

18. Información de interés.

Dentro de la carpeta del proyecto subida a GitHub se encuentra subidos los siguientes ficheros:

USUARIOS-ACCESOS.txt : este fichero contiene el usuario y su correspondiente clave para poder acceder al usuario cuando se pida el Login.

DISEÑO_BASEdeDATOS.txt :este es un fichero JPG donde aparece la estructura de la base de datos en estos momento. No está relacionada porque con Laravel daba muchos problemas y se requiere más tiempo para poder hacerlo de forma correcta.

login4.sql :es la base de datos que hemos implementado.

INFORME_Proyecto_Knowbuysell_VF.pdf: fichero donde guardamos este documento que estás leyendo.

19. Bibliografía

Libros:
Laravel. Curso práctico avanzado. José López Quijado
Laravel. Curso completo. Santiago Aguirre.

Web:
https://www.youtube.com/watch?v=LPrjJ29BQxI&t=113s
https://www.youtube.com/watch?v=n04ic-ALRvw
https://www.youtube.com/watch?v=j7bml8EQpIk

ChatGPT (para intentar localizar los errores del proyecto que no conseguí saber de dónde venían).