**Documentación del Proyecto PedidosJa**

1. **Introducción.**

**PedidosJa** es un sistema de gestión de pedidos grupales desarrollado para resolver el problema cotidiano en la oficina de la organización de pedidos entre varias personas, la asignación de responsabilidades y el cálculo de contribuciones. El proyecto nació como una idea en una oficina, donde los empleados solían pedir comida en grupo y necesitaban un sistema eficiente para coordinar el dinero y los pedidos.

Este proyecto fue concebido inicialmente como una forma de practicar y aplicar conocimientos en Laravel, y a medida que fue evolucionando, permitió abordar situaciones reales de la oficina con una solución digital.

El sistema utiliza **Laravel 11** tanto para el backend como para el frontend, y **Vue.js** para la interfaz, con **MySQL** como base de datos. Esta arquitectura facilita la interacción con los usuarios y la gestión de datos en tiempo real, ofreciendo una experiencia de usuario fluida y moderna.

Durante el desarrollo de **PedidosJa**, se aplicaron principios **SOLID** para garantizar que el código fuera mantenible, escalable y fácil de entender. Los principios SOLID ayudaron a estructurar el proyecto de manera modular, asegurando que las distintas funcionalidades fueran independientes y fáciles de modificar sin afectar el resto del sistema. Además, se optó por separar la lógica de negocio en servicios, lo que contribuyó a mantener los controladores enfocados solo en la gestión de las solicitudes HTTP.

El enfoque principal del sistema es la gestión de pedidos, permitiendo a los usuarios crear órdenes, agregar productos y llevar un control sobre el estado de cada pedido. A medida que el proyecto creció, se mejoraron las funcionalidades y se implementó una API RESTful, documentada con **Swagger**, para facilitar la interacción con los datos.

1. **Funcionalidades Principales.**

Este proyecto ofrece las siguientes funcionalidades:

* **Gestión de Órdenes:** Crear, actualizar y eliminar órdenes. Cada orden tiene un estado (por ejemplo, "borrador").
* **Gestión de Productos:** Los productos se registran con precios y descripciones, y pueden asignarse a las órdenes.
* **Gestión de Usuarios:** El sistema registra a los usuarios y permite asignarles roles para gestionar las órdenes.

### **Estructura del Sistema.**

El sistema de PedidosJa se basa en varias tablas principales que permiten gestionar las órdenes, los productos y los usuarios, relacionadas entre sí mediante claves foráneas. A continuación se detallan las tablas que forman la estructura principal del sistema:

* **Tabla users:**  
  Esta tabla almacena la información de los usuarios que interactúan con el sistema. Cada usuario tiene un *name* único que se guarda como una cadena de texto. Además, se gestionan las fechas de creación y actualización mediante los campos *created\_at* y *updated\_at.*
* **Tabla products:**  
  Registra los productos disponibles para los pedidos. Cada producto tiene un *name*, un *slug* (que se utiliza para las URLs amigables del producto) y un *reference\_price* que es el precio base del producto. Los campos *created\_at* y *updated\_at* gestionan las fechas de creación y actualización.
* **Tabla orders:**  
  Esta tabla contiene las órdenes realizadas, con un *reason* que describe la razón o el motivo de la orden. Cada orden está vinculada a un usuario encargado de la entrega mediante el campo *delivery\_user\_id*, que es una clave foránea a la tabla users. El campo order\_date almacena la fecha de la orden y state define el estado de la orden, que puede ser uno de los siguientes: '*draft*' (borrador), *'in\_process*' (en proceso), '*completed*' (completada), o '*canceled*' (cancelada).
* **Tabla order\_users:**  
  En esta tabla se almacena la información sobre los usuarios que participan en cada orden. Cada entrada está vinculada a una orden a través del campo *order\_id* (clave foránea a *orders*), y a un usuario mediante el campo *user\_id* (clave foránea a users). Además, se registra la cantidad de dinero entregado por el usuario en el campo *amount\_money*, así como el nombre del usuario (*user\_name*).
* **Tabla order\_user\_products:**  
  Esta tabla gestiona la relación entre los productos y los usuarios de la orden. Cada entrada está vinculada a un *order\_user\_id* (clave foránea a *order\_users*) y a un *product\_id* (clave foránea a products). Los campos adicionales incluyen el nombre del producto (*product\_name*), la cantidad pedida (quantity), una descripción adicional del producto (*description*) y el *final\_price*, que es el precio final del producto en esa orden.

**Otras Tablas:**  
Aunque existen otras tablas en la base de datos, no están directamente relacionadas con el flujo de datos principal y no afectan la gestión de órdenes, productos o usuarios. Estas tablas se utilizan para otros propósitos administrativos o de gestión interna y no se incluyen en el flujo operativo del sistema.

1. **Relaciones entre las tablas.**

El sistema PedidosJa utiliza las relaciones de Eloquent en Laravel para gestionar las interacciones entre las distintas entidades. A continuación, se describen las relaciones clave entre los modelos que corresponden a las tablas principales:

* Modelo Order:  
  El modelo Order tiene las siguientes relaciones:
  + deliveryUser: Una orden puede tener un usuario encargado de la entrega, lo que se modela como una relación *belongsTo* con el modelo User, utilizando el campo *delivery\_user\_id* como clave foránea.
  + orderUsers: Una orden puede tener múltiples usuarios asociados a través de la tabla intermedia *order\_users*. Esto se modela con una relación de tipo *hasMany*, lo que indica que una orden puede tener varios usuarios que participen en la misma.
* Modelo OrderUser:  
  El modelo OrderUser tiene las siguientes relaciones:
  + order: Un usuario está asociado a una orden específica. La relación es de tipo *belongsTo* con el modelo *Order*, utilizando el campo *order\_id* como clave foránea.
  + user: Cada *order\_user* está vinculado a un usuario en particular. Esto se modela como una relación de tipo *belongsTo* con el modelo User, utilizando el campo *user\_id* como clave foránea.
  + orderUserProducts: Un usuario puede haber solicitado varios productos en una orden. La relación es de tipo *hasMany* con el modelo *OrderUserProduct*, que representa los productos solicitados por ese usuario en la orden.
* Modelo OrderUserProduct:  
  El modelo OrderUserProduct tiene las siguientes relaciones:
  + orderUser: Un producto está asociado a un usuario en particular dentro de una orden. Esto se modela como una relación *belongsTo* con el modelo *OrderUser*, utilizando el campo *order user\_id* como clave foránea.
  + product: Cada producto en la tabla *order\_user\_products* está vinculado a un producto específico. La relación es de tipo *belongsTo* con el modelo *Product*, utilizando el campo *product\_id* como clave foránea.
* Modelo Product:  
  El modelo Product tiene la siguiente relación:
  + orderUserProducts: Un producto puede ser solicitado en múltiples órdenes. La relación es de tipo *hasMany* con el modelo *OrderUserProduct*, utilizando el campo *product\_id* como clave foránea.
* Modelo User:  
  El modelo User tiene las siguientes relaciones:
  + orderUsers: Un usuario puede participar en múltiples órdenes. La relación es de tipo *hasMany* con el modelo *OrderUser*, utilizando el campo *user\_id* como clave foránea.
  + orders: Un usuario puede crear múltiples órdenes. La relación es de tipo *hasMany* con el modelo *Order*, utilizando el campo *user\_id* como clave foránea.

En resumen:

* Una orden (Order) puede tener varios usuarios a través de la tabla *order\_users*.
* Un usuario puede estar asociado a varias órdenes y a varios productos en las órdenes mediante *order\_user\_products*.
* Un producto puede ser solicitado por varios usuarios dentro de diferentes órdenes.

1. **Inicialización del Proyecto.**

Una vez se tenga clonado el proyecto y abierto en un editor de texto:

* Instalar dependencia, en una terminal navegar a la dirección de la carpeta del proyecto y ejecutar el siguiente comando comando para instalar todas las dependencias de Composer: composer install
* Configurar el Archivo .env El siguiente paso es configurar el archivo .env, que contiene las variables de entorno necesarias para la aplicación. Copiar el archivo.env.exampley renombrarlo como.env**.**
* Una vez en .env configurar la conexión a la base de datos, por el nombre: DB\_DATABASE=pedidosja
* Generar la Clave de la Aplicación Laravel requiere una clave de aplicación única para funcionar correctamente. Se puede generar esta clave ejecutando el siguiente comando: php artisan key:generate
* Como ya se cuenta con la configuración correcta para la DB, ejecutar las migraciones y los seeders con el siguiente comando : php artisan migrate --seed

1. **Endpoints API.**

Se puede acceder a la documentación interactiva y completa de la API utilizando Swagger. Para probar los endpoints de la API de **PedidosJa**, seguir estos pasos después de hacer correr el proyecto:

1. **Inicia el proyecto Laravel** en tu entorno local: php artisan serve
2. **Accede a Swagger** a través de la ruta: /api/documentation.
3. **Explora y prueba** los diferentes endpoints directamente desde la interfaz de Swagger.

Esto te permitirá obtener información detallada sobre cada endpoint, los parámetros esperados, las respuestas y los códigos de estado. Esta documentación también proporciona ejemplos interactivos para probar la API directamente.

1. **Conclusión.**

Aunque **PedidosJa** es un proyecto relativamente sencillo, ha sido diseñado con una base flexible que permite futuras ampliaciones. Gracias a su estructura modular y al uso de buenas prácticas de desarrollo, como los principios **SOLID**, el sistema está preparado para crecer y mejorar con el tiempo. Existen varias áreas en las que se podrían implementar mejoras, como la integración de autenticación de usuarios, validaciones más complejas o nuevas funcionalidades como la gestión de pagos y notificaciones.

A medida que el proyecto avance, se seguirán explorando nuevas oportunidades para optimizar y añadir características, siempre con el objetivo de mejorar la **facilidad de uso**, la **gestión eficiente de los pedidos** y el **aprendizaje continuo** en el proceso de desarrollo. Este enfoque mantiene al sistema abierto a sugerencias y adaptaciones que puedan enriquecerlo en el futuro.