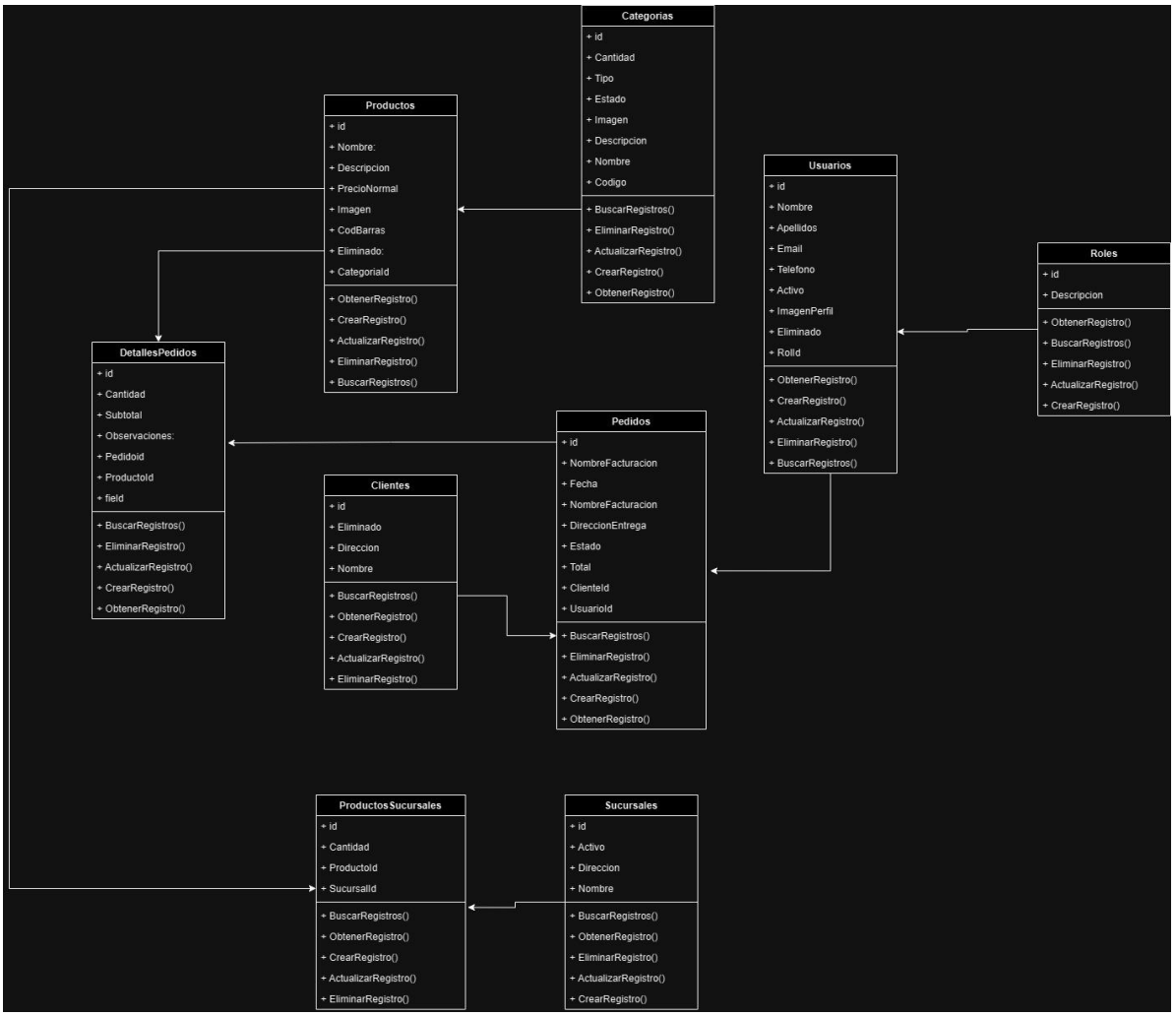
Diagrama de clases



Este diagrama de clases representa la estructura de un sistema de Tienda en Línea funcional. Se implementaron las tablas de Usuario para los distintos tipos de empleados dentro de la empresa (administrador y vendedor) y la tabla de Rol para definir estos tipos de usuario por separado y validar la información necesaria. La tabla de Producto nos sirve para ir revisando el catálogo de productos existentes en la tienda, así como la tabla Categoría que nos ayuda a tener un mejor orden y limpieza en los distintos productos. La tabla de Cliente nos ayuda a llevar un registro histórico de la gente que compra en la tienda. Por otra parte, la tabla de Sucursal nos permite validar las distintas localidades (si hubieran) físicas que tiene la empresa, además de validar la cantidad de producto existente por sucursal mediante la tabla de ProductoSucursal. Por último, un cliente al realizar un pedido se guarda en la tabla de Pedido y a la hora de pagar y utilizar el carrito de compras de la tienda, esta información se guarda dentro de la tabla DetallePedido para llevar un mejor orden y persistencia de la información ingresada. Los métodos nos permiten realizar operaciones de tipo CRUD esenciales para el manejo correcto de la información.

Cuadro comparativo

Característica a validar	Laravel	Symfony	CodeIgniter	PHP puro
Mantenimiento	Laravel facilita el mantenimiento con una estructura modular que sigue las mejores prácticas de desarrollo. La documentación es extensa y la comunidad está activa.	La estructura modular de Symfony facilita su mantenimiento, especialmente para proyectos grandes. La documentación es extensa y la comunidad brinda soporte.	La simplicidad de CodeIgniter hace que sea más fácil de mantener para proyectos más pequeños, pero los proyectos más grandes pueden ser más difíciles de mantener debido a la falta de algunas funciones avanzadas.	Mantener un proyecto en PHP puro puede resultar difícil a medida que se vuelve más complejo. La falta de una estructura definida puede dificultar el mantenimiento a largo plazo.
Seguridad	El software tiene un sistema de control de acceso integrado y seguro. Además, proporciona un mecanismo robusto que le permite manejar cualquier error o problema con facilidad. Además, el marco le permitirá almacenar contraseñas en un formato cifrado en lugar de texto, para que obtenga esa capa adicional de protección.	Symfony es conocido por su enfoque profesional de la seguridad. Proporciona componentes dedicados para autenticación, autorización y protección CSRF. Además, Symfony proporciona herramientas para reducir riesgos como la inyección de dependencias inseguras y la inyección de código.	CodeIgniter, aunque más ligero, incluye algunas características de seguridad, como la protección contra CSRF y funciones para escapar datos de salida y prevenir XSS. Sin embargo, comparado con Laravel y Symfony, puede requerir una configuración más manual para ciertas medidas de seguridad.	PHP puro no ofrece herramientas de seguridad integradas como marcos. La seguridad pura de PHP depende en gran medida de las prácticas de codificación segura de los desarrolladores. Si técnicas como la validación de entradas, la desinfección de datos y la prevención de inyección SQL no se implementan correctamente, son susceptibles a sufrir vulnerabilidades.
Escalabilidad	Los tamaños de los proyectos difieren. El	Escalabilidad horizontal: Al igual	Escalabilidad horizontal:	Escalabilidad Manual: PHP puro ofrece la

	software altamente escalable le permite abordar proyectos de cualquier tamaño, según sus necesidades. Laravel es uno de esos marcos escalables que facilita su uso en aplicaciones web pequeñas y medianas.	que Laravel, Symfony es altamente escalable horizontalmente debido a su arquitectura modular y modular. Puede distribuir la carga a otros servidores para manejar más tráfico. Flexibilidad: Symfony es conocido por su flexibilidad y capacidad para adaptarse a diversas necesidades, permitiendo cambios en los escenarios de desarrollo y expansión.	CodeIgniter es más liviano y una buena opción para proyectos pequeños, pero puede tener limitaciones en la parte horizontal en comparación con Laravel y Symfony. Simplicidad y rendimiento: CodeIgniter es más simple y puede funcionar mejor en proyectos más pequeños, pero esto puede resultar problemático a medida que aumenta la complejidad y el tamaño del proyecto.	máxima flexibilidad, pero la escalabilidad depende en gran medida de las habilidades del desarrollador y de la implementación de prácticas de codificación eficientes. No proporciona características integradas para la gestión de la escalabilidad, lo que puede requerir más esfuerzo manual.
Aprendizaje	Muchos desarrolladores web coinciden en que Laravel se encuentra entre los frameworks web más accesibles. Esto es gracias a la exhaustiva documentación de usuario que está disponible en la forma más sencilla. Además, el software incluye screencasts en PHP que son fáciles de entender.	Curva de aprendizaje: Symfony tiene una curva de aprendizaje más pronunciada en comparación con Laravel debido a su enfoque más sólido y más funciones. Sin embargo, esto significa que los desarrolladores sólo pueden aprender e implementar las funciones necesarias para un proyecto. Symfony tiene documentación detallada y una comunidad activa.	Curva de Aprendizaje: CodeIgniter es conocido por su simplicidad y, por lo tanto, tiene una curva de aprendizaje más baja en comparación con Laravel y Symfony. Su estructura liviana hace que sea fácil para los desarrolladores comenzar rápidamente y comprender los conceptos	Curva de Aprendizaje: PHP puro puede ser fácil de aprender para principiantes, pero la falta de estructuras y herramientas puede hacer que la curva de aprendizaje sea más empinada a medida que los proyectos crecen en complejidad. Facilitadores: No hay un marco específico para aprender en PHP puro, lo que puede ser tanto una ventaja como una desventaja. Los desarrolladores tienen la máxima flexibilidad, pero también deben

		Además, los proyectos de Symfony se pueden inicializar más fácilmente utilizando Flex, un sistema de configuración sencillo.	básicos del framework. Facilitadores: CodeIgniter tiene una documentación clara y un enfoque sencillo, lo que facilita la comprensión y la adopción. Sin embargo, su simplicidad puede ser una limitación para proyectos más grandes y complejos.	asumir más responsabilidades relacionadas con la arquitectura y las mejores prácticas por sí mismos.

Patrones de diseño

Los patrones de diseño (design patterns) son soluciones habituales a problemas comunes en el diseño de software. Cada patrón es como un plano que se puede personalizar para resolver un problema de diseño particular de tu código.

- Los patrones de diseño son un juego de herramientas de soluciones comprobadas a problemas habituales en el diseño de software. Incluso aunque nunca te encuentres con estos problemas, conocer los patrones sigue siendo de utilidad, porque te enseña a resolver todo tipo de problemas utilizando principios del diseño orientado a objetos.
- Los patrones de diseño definen un lenguaje común que puedes utilizar con tus compañeros de equipo para comunicaros de forma más eficiente. Podrías decir: “Oh, utiliza un singleton para eso”, y todos entenderían la idea de tu sugerencia. No habría necesidad de explicar qué es un singleton si conocen el patrón y su nombre.

Patrones creacionales

Los patrones creacionales proporcionan varios mecanismos de creación de objetos que incrementan la flexibilidad y la reutilización del código existente.

Factory Method: Es un patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.

Patrones Estructurales

Los patrones estructurales explican cómo ensamblar objetos y clases en estructuras más grandes, a la vez que se mantiene la flexibilidad y eficiencia de estas estructuras.

Adapter es un patrón de diseño estructural que permite la colaboración entre objetos con interfaces incompatibles.

Justificacion:

Estos patrones de diseño, en particular el método de fábrica y los adaptadores, se seleccionaron por sus beneficios únicos y su capacidad para abordar situaciones comunes en el diseño de software.

Factory Method:

Flexibilidad en la creación de objetos: El patrón Método de fábrica proporciona una interfaz para crear objetos en superclases y permite a las subclases cambiar el tipo de objeto que se crea.

Esto es especialmente útil cuando se trabaja con una familia de objetos relacionados, pero con diferentes implementaciones específicas.

La flexibilidad inherente de este patrón facilita la reutilización del código al permitirle adaptarse fácilmente a los requisitos cambiantes sin cambiar las estructuras existentes.

Abstracción de compilación: encapsular el proceso de compilación en una interfaz común proporciona un mayor nivel de abstracción.

Esto le permite cambiar la implementación concreta de la creación de objetos sin afectar el código del cliente, lo que facilita la gestión de la complejidad del código.

Adapter:

Colaboración entre objetos con interfaces incompatibles: El patrón del adaptador es importante cuando se trabaja con clases o componentes que tienen interfaces incompatibles.

Proporciona una manera para que objetos con diferentes interfaces trabajen juntos de manera efectiva.

Los adaptadores permiten la interoperabilidad entre componentes existentes sin cambiar el código fuente.

Esto es muy importante en situaciones en las que no es posible o deseable modificar el código fuente original.

Reutilización de códigos y sistemas existentes: los adaptadores facilitan la integración de sistemas existentes al proporcionar una interfaz común.

Esto es especialmente útil en el desarrollo de software, donde a menudo se trabaja con bibliotecas, clases y módulos existentes que no siguen la misma interfaz.

La capacidad de personalizar estas interfaces facilita la reutilización del código y ahorra tiempo y esfuerzo en el proceso de desarrollo.