Informe Técnico - Trabajo Práctico de Sistemas de Pedidos

Integrantes:

- 111136 Leonardo Dragone
- 109665 Valentin Calomino
- 103295 Francisco Zimbimbakis
- 107274 Lisandro Roman
- 97190 Franco Bragantini
- 108082 Celeste de Benedetto

Objetivo

El objetivo del presente trabajo práctico es diseñar e implementar un sistema que permita a los usuarios realizar pedidos de productos de forma dinámica y configurable, asegurando la extensibilidad de atributos, la validación de reglas de negocio y el correcto manejo de pedidos y stock.

Análisis del Problema

- Requisitos funcionales

  Lista de funciones clave:

  - ☐ Registro y autenticación con validación de email.
  - ☐ Recuperación de cuenta.
  - ☐ Gestión de productos (creación, atributos dinámicos, stock).
  - ☐ Creación, modificación y visualización de pedidos.
  - ☐ Aplicación de reglas dinámicas para validar pedidos.
  - ☐ Notificación de confirmación por correo electrónico.
  - ☐ Administración de pedidos por un backoffice.

- Requisitos no funcionales
  - ☐ Sistema extensible y configurable.
  - ☐ Autenticación segura con Basic Auth y JWT.
  - ☐ Persistencia confiable de datos con base de datos relacional.
  - ☐ UI básica funcional para probar el sistema.

Historias de Usuario

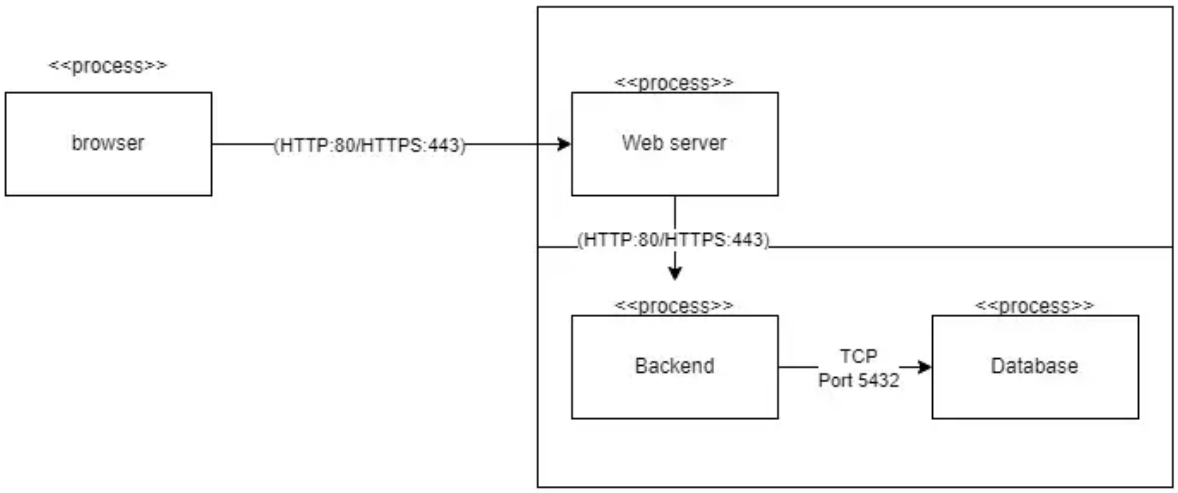
Como usuario del sistema **quiero** poder registrarme **para** darme de alta en el sistema	**Dado** que quiero registrarme **Cuando** ingreso email, contraseña, nombre, apellido, foto, edad, género y domicilio **Entonces** se crea mi cuenta y obtengo acceso al sistema
	Dado que quiero registrarme **Cuando** confirmo el formulario con un dato faltante **Entonces** obtengo un error 400 Bad Request indicando el campo faltante (ejemplo: "Nombre faltante").
Como usuario del sistema ya registrado **quiero** recuperar la contraseña **para** acceder a mi cuenta	**Dado** que me encuentro recuperando la contraseña **Cuando** ingreso mi email correctamente **Entonces** recibo un código de recuperación
	Dado que me encuentro recuperando la contraseña **Cuando** ingreso un email incorrecto **Entonces** obtengo un error 400 Bad Request
Como usuario ya registrado **Quiero** iniciar sesión **Para** acceder al sistema	**Dado** que quiero iniciar sesión **Cuando** ingreso email y contraseña correctos **Entonces** obtengo un token de autenticación
	Dado que quiero iniciar sesión **Cuando** ingreso email y/o contraseña incorrectos **Entonces** obtengo un error 400 Bad Request
Como usuario del sistema **Quiero** crear un pedido con productos **Para** seleccionar lo que deseo comprar	**Dado** que quiero crear un pedido **Cuando** selecciono los productos y sus características **Entonces** obtengo información del pedido creado
	Dado que quiero crear un pedido **Cuando** no selecciono ningún producto **Entonces** no puedo confirmar el pedido y recibo un error apropiado
Como usuario del sistema **Quiero** confirmar un pedido **Para** finalizar mi compra	**Dado** que quiero crear un pedido **Cuando** selecciono los productos **Entonces** puedo confirmarlo
	Dado que quiero crear un pedido **Cuando** no selecciono productos **Entonces** no puedo confirmarlo

Como usuario del sistema **Quiero** recibir un correo electrónico **Para** confirmar los artículos seleccionados	**Dado** que quiero recibir la confirmación del pedido **Cuando** el cliente ya confirmo el pedido **Entonces** recibo un mail de confirmación
Como usuario del sistema **Quiero** ver la lista de pedidos realizados **Para** tener control sobre ellos	**Dado** que quiero controlar mis pedidos **Cuando** ya he realizado pedidos **Entonces** recibo una lista de los pedidos anteriores
	Dado que quiero controlar mis pedidos **Cuando** no he realizado ningún pedido **Entonces** obtengo un bad request
Como usuario del sistema **quiero** cancelar un pedido **para** no recibirlo	**Dado** que quiero cancelar un pedido **Cuando** ya ha sido confirmado **Entonces** recibo confirmación de la cancelación si no han pasado 24h y está en PROCESO
	Dado que quiero cancelar un pedido **Cuando** ya ha sido confirmado **Entonces** recibo un bad request porque ya han pasado 24h o el pedido no está EN PROCESO
HISTORIAS DE USUARIO: ROL ADMIN	
Como administrador del sistema **quiero** crear productos con atributos correspondientes **para** generar un stock inicial	**Dado** que quiero crear un producto **Cuando** ingreso información correcta **Entonces** recibo un 201 CREATED
	Dado que quiero crear un producto **Cuando** ingreso información incorrecta **Entonces** obtengo un error **400 Bad Request**
Como administrador del sistema **Quiero** agregar atributos a productos existentes **Para** configurarlos según sea necesario	**Dado** que quiero agregar atributos a un producto **Cuando** ingreso información correcta **Entonces** el producto se actualiza con los nuevos atributos
	Dado que quiero agregar atributos a un producto **Cuando** ingreso información incorrecta **Entonces** obtengo un error 400 Bad Request

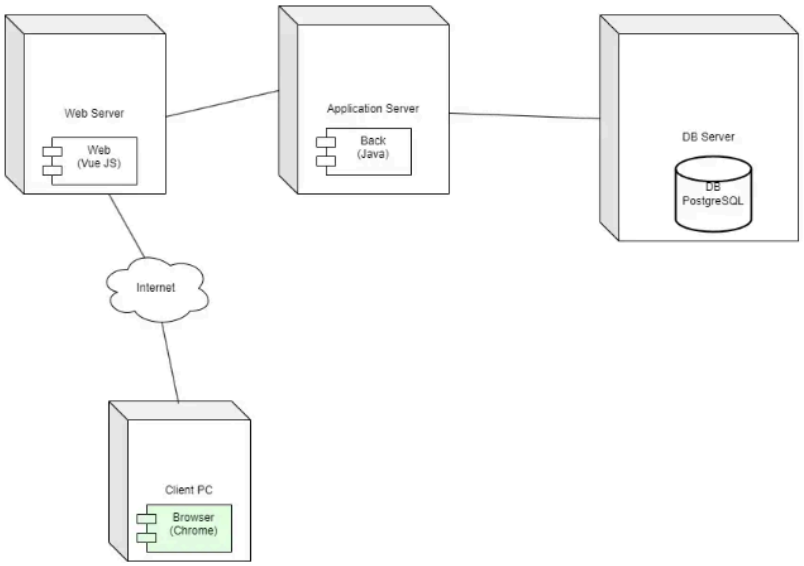
Como administrador del sistema **Quiero** incrementar el stock de un producto **Para** mantener la disponibilidad	**Dado** que quiero incrementar el stock **Cuando** ingreso los valores correctos **Entonces** recibo la información actualizada
Como administrador del sistema **quiero** observar los pedidos confirmados **para** administrarlos	**Dado** que quiero ver los pedidos confirmados **Cuando** existen pedidos **Entonces** recibo una lista con los detalles
	Dado que quiero ver los pedidos confirmados **Cuando** no hay pedidos **Entonces** obtengo un error `404 Not Found`
Como administrador del sistema **quiero** modificar el estado de un pedido **para** notificar su estado actual	**Dado** que quiero modificar el estado de un pedido **Cuando** el pedido existe **Entonces** recibo la información actualizada con el nuevo estado

Diseño de Arquitectura (Modelo 4+1)

- ### Vista de Procesos

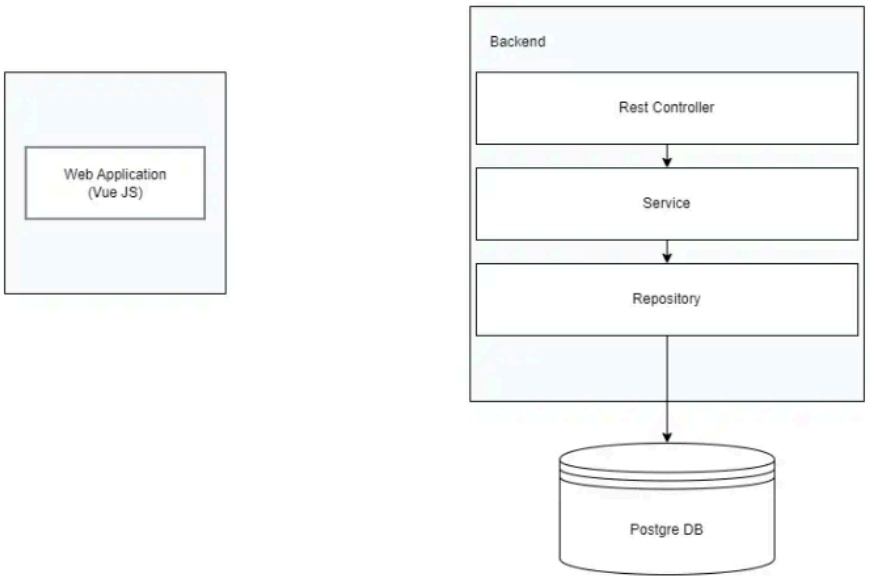


- ### Vista de Desarrollo

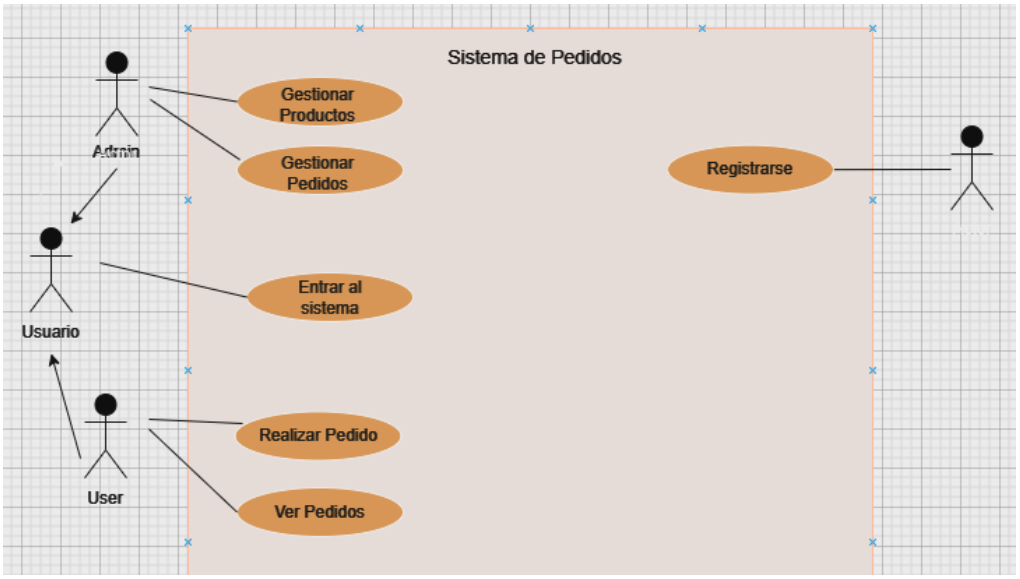


- **Vista Física**

- **Vista Lógica**



- **Vista de Escenarios**



Estrategia de Desarrollo

- **Metodología**:
  Usaremos Scrum con 4 sprints de 1 semana.
  - Sprint 1: Implementar registro y autenticación.
  - Sprint 2: Gestión de productos y atributos dinámicos.
  - Sprint 3: Validación de reglas de negocio y pedidos.
  - Sprint 4: Mejoras finales.
- **Herramientas**:
  - JIRA: Gestión de historias y tareas.
  - GitLab: Control de versiones.
  - Docker: Entorno aislado para desarrollo y despliegue.

Desarrollo del Backend

Nuestro sistema permite crear órdenes de productos, aplicar reglas de negocio personalizadas a las órdenes, y gestionar su estado a lo largo de un flujo de trabajo definido.

Para lograr esto, se ha implementado un modelo de datos que abarca varias entidades clave como Order, Product, OrderProduct, OrderStatus, y User. Cada una de estas entidades interactúa entre sí para facilitar la creación de órdenes, la validación de productos y la actualización del estado de las órdenes.

El sistema ha sido diseñado para ser flexible y escalable, lo que permite agregar fácilmente nuevas reglas de negocio sin necesidad de modificar el código base. Las reglas están encapsuladas en una jerarquía de clases que heredan de OrderRule, y el uso del patrón **Single Table Inheritance** facilita la extensión de las reglas sin comprometer la estructura general.

Se emplean tecnologías como **Spring Boot** para la construcción del backend, **JPA** para la persistencia de datos. Además, el diseño modular de las reglas de negocio asegura que el sistema pueda adaptarse a cambios futuros en los requisitos de negocio de manera eficiente.

Entidades y Relaciones

El sistema está compuesto por las siguientes entidades principales:

- **Order:** Representa una orden de compra realizada por un usuario. Tiene una relación de uno a muchos con OrderProduct, lo que significa que cada orden puede contener múltiples productos. Además, cada orden tiene un estado asociado (OrderStatus), que puede ser CREATED, CONFIRMED, IN_PROCESS, o SENT.
- **OrderProduct:** Representa los productos dentro de una orden. Cada OrderProduct tiene atributos adicionales que describen las características específicas del producto, como el nombre del producto, el ID del producto y la cantidad. Además, esta entidad tiene un método para verificar si un producto tiene un atributo específico, lo que facilita la validación de reglas personalizadas.

- **User:** Representa al usuario del sistema. La relación de uno a muchos con Order permite que un usuario pueda tener múltiples órdenes. El usuario está asociado con detalles como su nombre, dirección, correo electrónico, etc. Además, implementa la interfaz UserDetails de Spring Security, lo que facilita la autenticación y autorización.
- **Product:** Define los productos disponibles en la tienda. Esta entidad también tiene atributos específicos y una relación de muchos a muchos con OrderProduct, ya que un producto puede estar presente en múltiples órdenes.

Reglas de Orden (OrderRules)

El sistema permite la creación de reglas de negocio personalizadas que se aplican a las órdenes. Estas reglas están representadas por la clase abstracta OrderRule y se implementan a través de diferentes fábricas de reglas, como AndRuleFactory, OrRuleFactory, NotRuleFactory, etc. Cada regla es una instancia de OrderRule, que puede evaluar si una orden cumple o no con las condiciones específicas.

Las fábricas de reglas permiten crear dinámicamente nuevas instancias de reglas sin necesidad de modificar el código base, simplemente agregando nuevas fábricas. Este enfoque facilita la extensibilidad del sistema y la flexibilidad en el diseño de las reglas de negocio.

Persistencia de Datos

Para la persistencia de datos, se utiliza JPA (Java Persistence API) junto con la anotación @Entity para definir las entidades. La relación entre las entidades Order y OrderProduct se maneja mediante la anotación @ManyToOne y @OneToMany, respectivamente, mientras que @ElementCollection se utiliza para almacenar atributos complejos como los productos y las órdenes.

Flujo de Trabajo

1. Un usuario realiza una orden que contiene varios productos.
2. Cada producto en la orden puede tener un conjunto de atributos específicos, que se almacenan como un mapa de atributos.

3. Las reglas de negocio se aplican a la orden a través de las fábricas de reglas. Por ejemplo, se pueden crear reglas que validen el número de productos o el total de ciertos atributos.
4. El sistema verifica si la orden cumple con las reglas definidas y actualiza el estado de la orden (por ejemplo, de CREATED a CONFIRMED).

Decisiones clave de la arquitectura

1. Uso de una arquitectura de capas (Layered Architecture). En esta se dividen las responsabilidades de cada capa, siendo las principales:
    a. Controller: Se encarga de la comunicación entre el cliente y la aplicación. Son las clases que manejan las solicitudes HTTP, recibir las peticiones de los usuarios, procesarlas y devolver las respuestas.
    b. Service: Se encarga de procesar los pedidos que recibe el controller. Aquí se gestionan las operaciones que la aplicación realiza en los datos, como la creación de un pedido o la gestión del stock.
    c. Repository: Es la capa encargada de la comunicación con la base de datos. Son los repositorios que gestionan las entidades correspondientes en la base de datos, permitiendo realizar operaciones de persistencia y consulta sobre ellas
2. Patrón de Fábrica: Las fábricas de reglas (RuleFactory) son clave para garantizar que el sistema sea extensible. Nuevas reglas pueden añadirse simplemente creando nuevas clases de fábricas, sin necesidad de tocar la lógica existente.
3. Modularidad y Extensibilidad: Al separar las responsabilidades (creación de reglas, persistencia de datos, validación de ordenes, etc.) y utilizar patrones de diseño como el de fábrica, el sistema es fácil de mantener y extender.

Se configuró un proyecto en **Postman** para probar y validar los endpoints del backend, asegurando su correcto funcionamiento con el frontend. Este entorno permite realizar pruebas automatizadas de las

API y documentar las respuestas de los endpoints para facilitar la integración. Ingresar al siguiente link:

https://tp-ing-software.postman.co/workspace/Tp.-Ing.-Software-Workspace~0576c123-6c30-42a7-8503-3d3378e031d6/collection/39273604-82d8bad0-256b-4a73-b7f0-57890b76de3f?action=share&creator=39320640

Desarrollo del Frontend

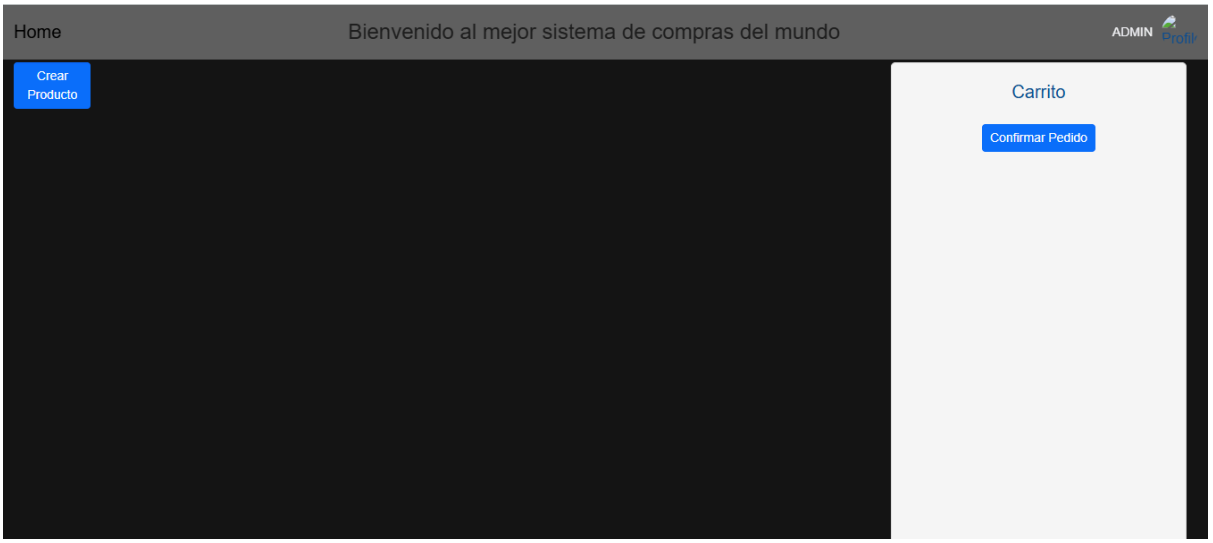
El frontend del proyecto está desarrollado utilizando Vue.js, un framework progresivo de JavaScript que permite crear interfaces de usuario interactivas y reactivas. Vue.js es conocido por su fácil integración, su enfoque basado en componentes, y su capacidad para crear aplicaciones dinámicas de una manera eficiente.

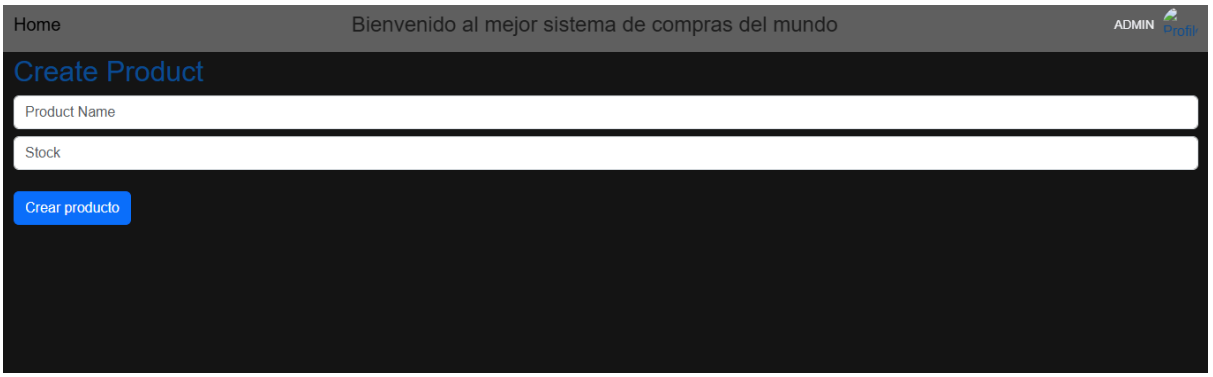
Cómo Ejecutar el Proyecto:

Para ejecutar el frontend en un entorno local, basta con seguir estos pasos:

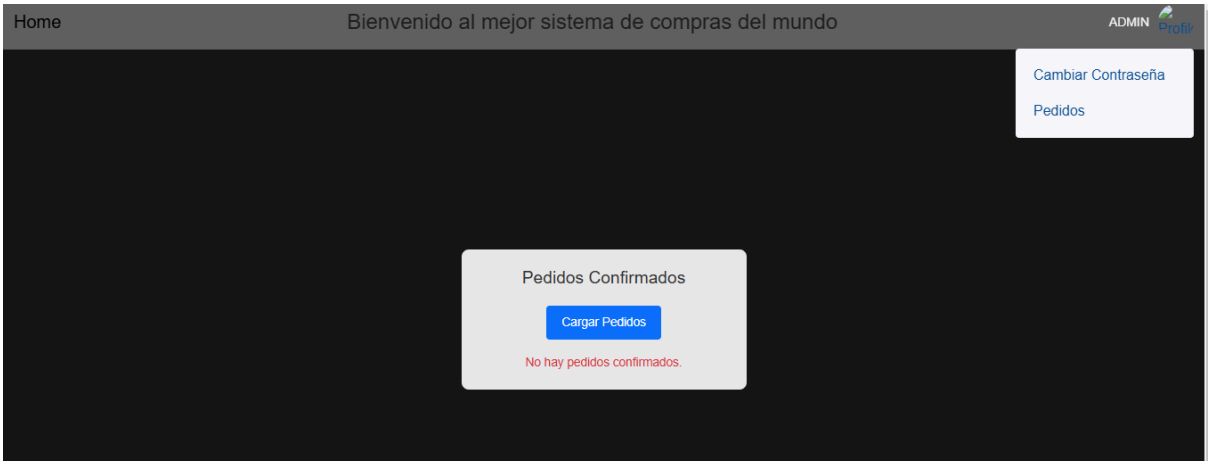
1. **Instalar Dependencias**: Ejecutar npm install en la terminal para instalar todas las dependencias del proyecto.
2. **Iniciar el Servidor**: Utilizar el comando npm run serve para iniciar el servidor de desarrollo en http://localhost:8080, donde se podrá visualizar la aplicación.

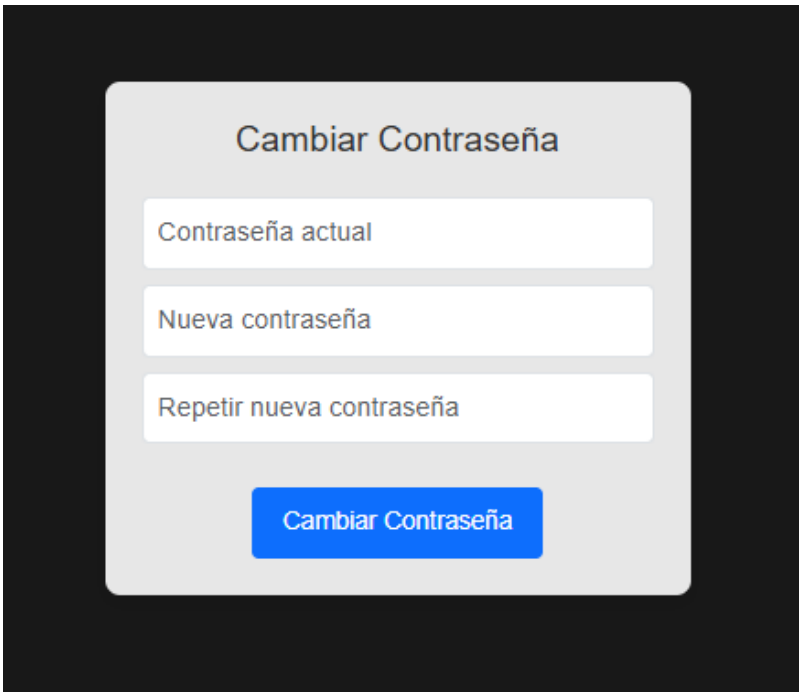
El desarrollo de este frontend está orientado a ofrecer una experiencia de usuario fluida y rápida, aprovechando las capacidades de Vue.js, Vuex y Axios para una comunicación eficiente con el backend y una interfaz de usuario atractiva gracias a Bootstrap.


Home
Bienvenido al mejor sistema de compras del mundo
ADMIN
Crear Producto
Carrito
Confirmar Pedido


Home
Bienvenido al mejor sistema de compras del mundo
ADMIN
Create Product
Product Name
Stock
Crear producto


Order Summary
Clear Cart
Accept Order


Home
Bienvenido al mejor sistema de compras del mundo
ADMIN
Cambiar Contraseña
Pedidos
Pedidos Confirmados
Cargar Pedidos
No hay pedidos confirmados.


Cambiar Contraseña
Contraseña actual
Nueva contraseña
Repetir nueva contraseña
Cambiar Contraseña