**E-BUSINESS-FERRETERÍA**

**DOCUMENTACIÓN DE DISEÑO, PRUEBAS Y MANTENIMIENTO**

Fundamentos de Ingeniería de Software



Eduardo Pineda Almeda (193109)

Carlos Murillo Flores (202179)

Gabriel Alberto Vilchis Ríos (215460)

Emmanuel Rodríguez Franco (215822)

**7/11/24 - 19/11/24**

**INDICE**

**Diseño**

Elicitación de requisitos--------------------------------------------------------------------------------1

Formalización de requisitos----------------------------------------------------------------------------2

Lista de productos inicial-------------------------------------------------------------------------------4

Diagrama de flujo de datos-----------------------------------------------------------------------------5

Definición de patrón Modelo Vista Controlador----------------------------------------------------9

Arquitectura de software------------------------------------------------------------------------------10

Diagrama de clases------------------------------------------------------------------------------------13

Clasificación de productos de catálogo-------------------------------------------------------------15

Diagrama de entidad relación------------------------------------------------------------------------17

Diagrama lógico relacional---------------------------------------------------------------------------18

Diseño de tablas----------------------------------------------------------------------------------------20

Implementación de base de datos--------------------------------------------------------------------21

Implementación de interfaz--------------------------------------------------------------------------23

**Pruebas**

Pruebas unitarias---------------------------------------------------------------------------------------33

Pruebas de integración--------------------------------------------------------------------------------34

Pruebas funcionales-----------------------------------------------------------------------------------34

Pruebas de interfaz------------------------------------------------------------------------------------34

Pruebas de rendimiento-------------------------------------------------------------------------------34

**Mantenimiento**

Mantenimiento correctivo----------------------------------------------------------------------------35

Mantenimiento preventivo---------------------------------------------------------------------------35

Mantenimiento perfectivo----------------------------------------------------------------------------36

Mantenimiento adaptativo----------------------------------------------------------------------------36

**ELICITACIÓN DE REQUISITOS**

Se recopilaron los requisitos que solicitaba el usuario final para la elaboración de un e-business de una ferretería en la cual se solicitó que se cumpliera con lo siguiente:

* Permitir a los clientes registrarse y autenticarse.
* Mostrar los productos de la ferretería con sus descripciones, precios y disponibilidad.
* Facilitar la selección y compra de productos.
* Implementar un sistema de pagos en línea.
* Para el administrador, gestión y actualización del stock.
* Enviar confirmación de pedido y notificaciones de entrega.
* Permitir a los usuarios consultar sus pedidos anteriores.
* Facilitar comentarios de usuarios sobre productos.

**1**

**FORMALIZACIÓN DE REQUISITOS**

Una vez obtenidos los requisitos se clasificaron y formalizaron según el tipo de requisito, ya sea funcional o no funcional, dándoles un identificador, un nombre clave, una descripción y definiendo el nivel de prioridad.

|  |  |
| --- | --- |
| **Identificaciones del requerimiento:** | REQ-1 |
| **Nombre del requerimiento:** | Registro y autenticación de usuarios |
| **Características:** | Registro y autenticación del usuario. |
| **Descripciones del requerimiento:** | Se podrá iniciar sesión y registro del usuario, donde este deberá ingresar su nombre y apellido, correo electrónico, numero telefónico y dirección. |
| **Prioridad** | Alta |

|  |  |
| --- | --- |
| **Identificaciones del requerimiento:** | REQ-2 |
| **Nombre del requerimiento:** | Catálogo de productos |
| **Características:** | Navegación y visualización del catálogo. |
| **Descripciones del requerimiento:** | Se podrá navegar productos en distintos donde estarán con su información como lo son la descripción del producto, el precio y disponibilidad del producto. |
| **Prioridad** | Alta |

|  |  |
| --- | --- |
| **Identificaciones del requerimiento:** | REQ-3 |
| **Nombre del requerimiento:** | Carrito de compras |
| **Características:** | Selección y compra de productos. |
| **Descripciones del requerimiento:** | Se podrá seleccionar y comprar productos del catálogo, estos serán almacenados en un carrito de compra que será de utilidad para confirmar el pago de los productos solicitados por el cliente. |
| **Prioridad** | Alta |

**2**

|  |  |
| --- | --- |
| **Identificaciones del requerimiento:** | REQ-4 |
| **Nombre del requerimiento:** | Sistema de pagos |
| **Características:** | Sistema de transacciones |
| **Descripciones del requerimiento:** | Se podrá realizar pagos de los pedidos en línea. |
| **Prioridad** | Alta |

|  |  |
| --- | --- |
| **Identificaciones del requerimiento:** | REQ-5 |
| **Nombre del requerimiento:** | Gestión de inventario |
| **Características:** | Registro de productos |
| **Descripciones del requerimiento:** | Se podrá registrar productos en el inventario, incluyendo detalles como el nombre, precio, tipo de categoría, cantidad disponible en la tienda. Así como actualizar la información de los productos existentes, como el precio y la cantidad en stock. Y en caso de que un producto se agote o se tenga que deshabilitar por alguna razón, también podrá realizar esa acción. |
| **Prioridad** | Alta |

|  |  |
| --- | --- |
| **Identificaciones del requerimiento:** | REQ-6 |
| **Nombre del requerimiento:** | Notificaciones |
| **Características:** | Corroboración de pedidos |
| **Descripciones del requerimiento:** | Se podrá mandar notificaciones al usuario, con la intención de confirmar al usuario que se fue confirmado su pedido, y de avisar cuando será el estimado de la entrega del producto solicitado. |
| **Prioridad** | Media |

|  |  |
| --- | --- |
| **Identificaciones del requerimiento:** | REQ-7 |
| **Nombre del requerimiento:** | Historial de compras |
| **Características:** | Visualización de historial |
| **Descripciones del requerimiento:** | Se podrá ver el historial de los pedidos realizados por el usuario, esto incluye pedidos anteriores, así como la fecha en la que se realizó el pedido, y el respectivo monto que pagó el usuario. |
| **Prioridad** | Media |

**3**

A los requisitos obtenidos y formalizados anteriormente, se le clasificaron como funcionales, y de igual forma se incluyeron los requisitos no funcionales que son de suma importancia para definir ciertos comportamientos del sistema:

* El sistema debe ser capaz de manejar diversos usuarios concurrentes realizando operaciones como registro, navegación en el catálogo, y pagos sin comprometer la velocidad de respuesta. Dicha velocidad de respuesta no debe exceder de los 2 segundos para cualquier acción que realicen los usuarios.
* El sistema debe garantizar la seguridad de los datos personales y de pago mediante el cifrado de la información sensible, como contraseñas y datos bancarios, utilizando un protocolo seguro como HTTPS. Este cifrado debe confirmar que no existan vulnerabilidades críticas en el manejo de datos sensibles.
* El sistema debe ser compatible con navegadores web más populares, como lo son Google Chrome, Brave, Microsoft Edge, Safari, Opera GX y Mozilla Firebox.
* El sistema debe ser escalable, permitiendo agregar más productos al catálogo, usuarios registrados y capacidad de procesamiento de transacciones sin necesidad de rediseño del sistema.
* El sistema debe estar diseñado de forma modular, permitiendo la adición de nuevas funcionalidades o la corrección de errores con un impacto mínimo en las funcionalidades existentes.
* El sistema debe ser intuitivo y fácil de usar, permitiendo que el usuario pueda completar tareas como el registro, navegación en el catálogo y la compra en menos de 5 minutos sin requerir soporte técnico.
* El sistema debe estar disponible para los usuarios al menos en un 99.9% del tiempo, exceptuando tiempos que involucren mantenimientos planificados previamente.

**LISTA DE PRODUCTOS INICIAL**

Se definió un listado inicial de productos que conformaría el catálogo que la ferretería dispondría, donde se incluyeron productos comunes dentro del campo con un precio estimado.

* Destornillador plano, mediano – $50 MXN
* Destornillador Phillips, grande – $60 MXN
* Llave inglesa de 6" – $120 MXN
* Llave inglesa de 8" – $150 MXN
* Llave inglesa de 12" – $200 MXN
* Alicate universal, mediano – $120 MXN
* Alicate de corte, pequeño – $80 MXN
* Alicate de presión, grande – $250 MXN
* Martillo de carpintero, 400g – $90 MXN
* Martillo de goma, 300g – $70 MXN
* Cinta métrica de 5m – $40 MXN
* Cinta métrica de 10m – $70 MXN
* Cinta métrica de 25m – $200 MXN
* Cinta métrica láser, 50m – $800 MXN
* Serrucho para madera, 18" – $150 MXN

**4**

* Serrucho para metal, 12" – $120 MXN
* Cuchillo multiusos, retráctil – $50 MXN
* Cuchillo de hoja fija, 25mm – $80 MXN
* Tijeras de podar, pequeñas – $150 MXN
* Tijeras de podar, grandes – $250 MXN
* Brocas para madera, 3mm – $25 MXN
* Brocas para metal, 5mm – $30 MXN
* Brocas para concreto, 10mm – $40 MXN
* Clavos de acero, 1" (paquete de 100) – $30 MXN
* Tornillos para madera, 2" (paquete de 200) – $80 MXN
* Pegamento universal, 100g – $40 MXN
* Super glue, 20g – $25 MXN
* Silicona en tubo, 300ml – $100 MXN
* Bombilla LED 10W, blanca fría – $50 MXN
* Extensión eléctrica, 5m, 3 tomas – $150 MXN
* Cinta aislante negra, 19mm x 10m – $30 MXN
* Pintura en spray blanca, 400ml – $90 MXN
* Lija para madera, grano 80 – $40 MXN
* Lija para metal, grano 120 – $40 MXN

**Diagrama de Flujo de Datos (DFD) para Sistema de Ferretería**

**Entidades Externas**

**Usuario**

* **Descripción**: Representa al cliente o usuario que interactúa con el sistema. El usuario puede ser tanto un visitante que realiza compras sin registrarse como un usuario registrado.
* **Interacción**: El **Usuario** interactúa principalmente con el sistema en dos formas:
  + **Registro/Login**: El usuario puede registrar una cuenta o iniciar sesión para guardar su información de compra.
  + **Carrito de Compras**: El usuario puede agregar productos al carrito de compras, visualizar los productos disponibles, y proceder al pago.
  + **Pago**: El usuario puede realizar el pago para completar su compra.
* **Flujos**:
  + **Usuario → Registro de Usuario**: El usuario envía sus datos (nombre, correo, contraseña) al proceso de **Registro de Usuario** para ser almacenados en la **Base de Datos de Usuarios**.
  + **Usuario → Carrito de Compras**: El usuario agrega productos al carrito de compras, consulta el inventario y procede a hacer una compra.

**5**

* + **Usuario → Procesar Pago**: Después de llenar el carrito, el usuario finaliza la compra mediante el pago.

**Procesos**

**Registro de Usuario**

* **Descripción**: Este proceso se encarga de la creación de cuentas de usuario o la validación del login para usuarios ya existentes.
* **Interacción**: Recibe los datos de registro del **Usuario** y los almacena en la **Base de Datos de Usuarios**.
* **Flujos**:
  + **Usuario → Registro de Usuario**: El usuario envía sus datos (nombre, correo, contraseña).
  + **Registro de Usuario → Base de Datos de Usuarios**: Los datos del usuario se almacenan en la base de datos para su posterior acceso y autenticación.

**Carrito de Compras**

* **Descripción**: El **Carrito de Compras** gestiona los productos seleccionados por el usuario para su compra, calculando el total y la disponibilidad.
* **Interacción**: Consulta los productos disponibles en la **Base de Datos de Productos** y permite al **Usuario** agregar productos al carrito.
* **Flujos**:
  + **Usuario → Carrito de Compras**: El usuario selecciona y agrega productos al carrito.
  + **Carrito de Compras → Base de Datos de Productos**: El carrito consulta la base de datos de productos para obtener la disponibilidad y precios de los artículos.

**Procesar Pago**

* **Descripción**: Este proceso maneja el pago de la compra, gestionando la interacción con el **Proveedor de Pago** para verificar y procesar la transacción.
* **Interacción**: Recibe los detalles del pedido desde el **Carrito de Compras**, procesa el pago y actualiza el inventario y el registro de pedidos.
* **Flujos**:
  + **Carrito de Compras → Procesar Pago**: El carrito envía los productos seleccionados y el total de la compra al proceso de pago.
  + **Procesar Pago → Proveedor de Pago**: El proceso de pago se comunica con el proveedor de pago para verificar la autenticidad y autorizar la transacción.
  + **Procesar Pago → Base de Datos de Productos (Actualización)**: Si el pago es exitoso, el sistema actualiza la **Base de Datos de Productos** reduciendo la cantidad de productos en stock.
  + **Procesar Pago → Base de Datos de Pedidos**: Se registra la compra en la **Base de Datos de Pedidos**, creando un historial de la transacción.

**6**

**Almacenes de Datos (Tablas de Base de Datos)**

**Base de Datos de Usuarios**

* **Descripción**: Almacena la información de los usuarios registrados en el sistema, como su nombre, correo electrónico, contraseñas (debidamente cifradas) y otros detalles relacionados.
* **Interacción**: Se actualiza cada vez que un nuevo usuario se registra o inicia sesión.
* **Flujos**:
  + **Registro de Usuario → Base de Datos de Usuarios**: Los datos del usuario son almacenados en la base de datos.

**Base de Datos de Productos**

* **Descripción**: Contiene información detallada sobre los productos disponibles en la ferretería, como nombre, descripción, precio, cantidad en stock y otros atributos importantes.
* **Interacción**: El sistema consulta esta base de datos para obtener detalles de productos y actualiza la cantidad disponible cuando se realiza una compra.
* **Flujos**:
  + **Carrito de Compras → Base de Datos de Productos**: El carrito consulta la base de datos para obtener la información de los productos.
  + **Procesar Pago → Base de Datos de Productos (Actualización)**: La base de datos se actualiza después de una compra para reflejar la nueva cantidad de productos disponibles.

**Base de Datos de Pedidos**

* **Descripción**: Almacena todos los pedidos realizados por los usuarios, incluyendo los productos comprados, cantidades, precios, estado del pedido y el usuario asociado.
* **Interacción**: Se actualiza cada vez que un usuario completa una compra y se genera un nuevo pedido.
* **Flujos**:
  + **Procesar Pago → Base de Datos de Pedidos**: Después de un pago exitoso, se guarda la información del pedido en esta base de datos.

**Entidades Externas**

**Proveedor de Pago**

* **Descripción**: Es un sistema externo con el que el sistema interactúa para procesar los pagos de los usuarios. Este puede ser, por ejemplo, un sistema de pagos como Stripe, PayPal o un servicio bancario.
* **Interacción**: El sistema envía la información del pago al proveedor para su validación y autorización.

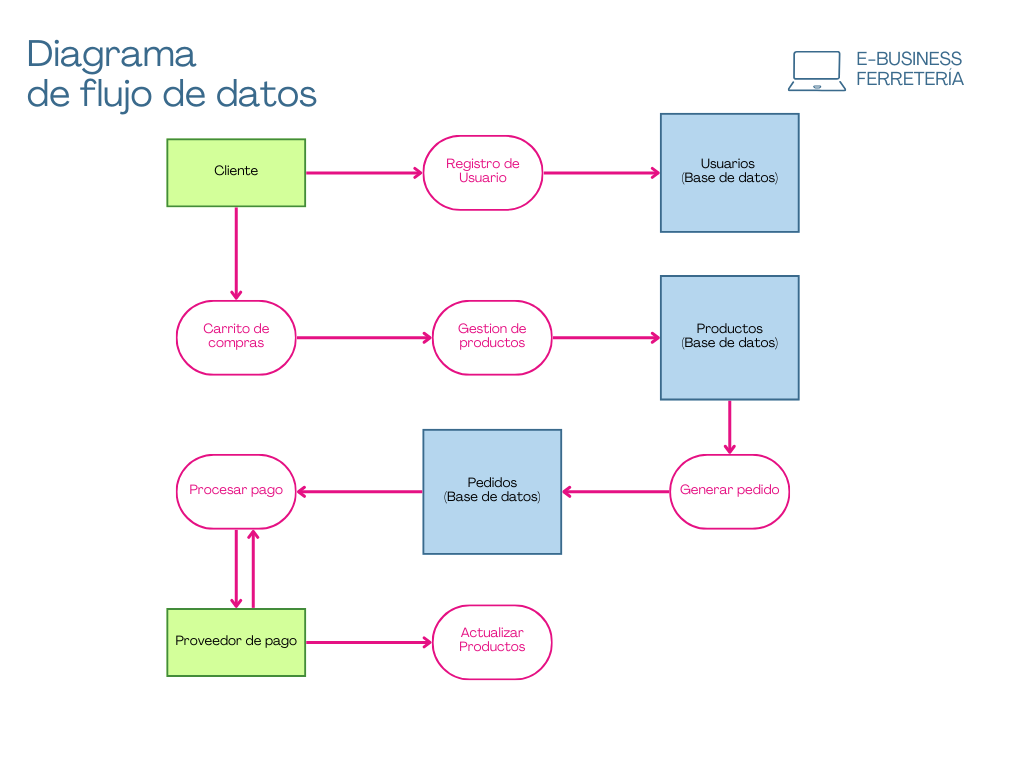
**7**

* **Flujos**:
  + **Procesar Pago → Proveedor de Pago**: El proceso de pago comunica los detalles de la transacción (monto, información del usuario) al proveedor de pago para autorizar la transacción.
  + **Proveedor de Pago → Procesar Pago**: El proveedor de pago devuelve la confirmación del estado de la transacción (éxito o fallo).

**Flujos de Datos**

* **Usuario → Registro de Usuario**: El flujo de datos del usuario hacia el proceso de registro incluye la información de registro (nombre, correo, contraseña).
* **Usuario → Carrito de Compras**: Los productos seleccionados por el usuario son enviados al carrito de compras.
* **Carrito de Compras → Base de Datos de Productos**: El carrito consulta la base de datos de productos para obtener los detalles de los artículos (disponibilidad, precios, etc.).
* **Carrito de Compras → Procesar Pago**: El carrito envía los productos y el total de la compra al proceso de pago.
* **Procesar Pago → Proveedor de Pago**: El proceso de pago interactúa con el proveedor de pago para verificar y autorizar la transacción.
* **Procesar Pago → Base de Datos de Productos (Actualización)**: Si el pago es exitoso, la base de datos de productos se actualiza, reduciendo la cantidad de productos en inventario.

**8**

* **Procesar Pago → Base de Datos de Pedidos**: Los detalles del pedido se registran en la base de datos de pedidos.

**9**

**DEFINICIÓN DE PATRÓN MODELO-VISTA-CONTROLADOR**

Para la arquitectura de software, se seleccionó un patrón de diseño MVC (Modelo, Vista, Controlador) para una aplicación web, utilizando las siguientes tecnologías y componentes:

El componente modelo se encarga de gestionar los datos y la lógica de negocio de la aplicación, el modelo está compuesto por las siguientes tecnologías:

* Como backend se utiliza Django como framework para la creación de la API RESTful, el cual, va a interactuar con la base de dichos.
* La base de datos se usará MySQL como herramienta para almacenar y gestionar la información del modelo.

En el modelo, realiza las siguientes acciones:

* El modelo contiene las definiciones de las tablas y relación de la base de datos, mediante el ORM que proporciona Django.
* El modelo realiza operaciones CRUD (Create, Read, Update, Delete), que consisten en la creación, la lectura, la actualización y la eliminación de datos.
* Y por último, el modelo proporciona datos procesados al controlador para que posteriormente sean enviados a la vista.

La vista es el componente responsable de mostrar la interfaz de usuario (UI) y mostrar la información de una manera que sea entendible al usuario. La vista esta compuesta por las siguientes tecnologías:

**10**

**ARQUITECTURA DE SOFTWARE**

Con base al diseño Modelo Vista Controlador (MVC) definido anteriormente, se realizó un diagrama de arquitectura de software donde se indica el flujo de datos y funcionamiento.

Diagrama

Descripción generada automáticamente

El componente modelo se encarga de gestionar los datos y la lógica de negocio de la aplicación, el modelo está compuesto por las siguientes tecnologías:

* Como backend se utiliza Django como framework para la creación de la API RESTful, el cual, va a interactuar con la base de dichos.
* La base de datos se usará MySQL como herramienta para almacenar y gestionar la información del modelo.

En el modelo, realiza las siguientes acciones:

* El modelo contiene las definiciones de las tablas y relación de la base de datos, mediante el ORM que proporciona Django.
* El modelo realiza operaciones CRUD (Create, Read, Update, Delete), que consisten en la creación, la lectura, la actualización y la eliminación de datos.
* Y por último, el modelo proporciona datos procesados al controlador para que posteriormente sean enviados a la vista.

La vista es el componente responsable de mostrar la interfaz de usuario (UI) y mostrar la información de una manera que sea entendible al usuario. La vista esta compuesta por las siguientes tecnologías:

**11**

* Como frontend, se seleccionó React como framework para gestionar la UI, en conjunto a HTML, CSS y JavaScript para el diseño y el comportamiento de la aplicación.

En la vista, realiza las siguientes acciones:

* La vista recibe los datos que fueron procesados desde el backend, que fue gracias al controlador.
* La vista muestra los datos en la interfaz gráfica.
* Por último, captura las acciones que realiza el usuario, como podrían ser los clics o interacciones con formularios, para enviarlos al controlador.

El componente del controlador actúa como un intermediario entre la vista y el modelo, esta constituido en 2 partes:

* El controlador del backend, este, esta representado por los endpoints de la API RESTful de Django, que ayuda a recibir las solicitudes que provienen de la vista, interactúan con el modelo para obtener o actualizar los datos, para posteriormente retornar los resultados en formato JSON a la vista.
* El controlador del frontend, eta representado por los componentes que proporciona React, los cuales, ayudan a manejar los eventos del usuario (las interacciones del usuario), realizar solicitudes con el protocolo HTTP (Hypertext Transfer Protocol) al backend, y actualizar el estado de la aplicaciones para mostrar los datos en la vista.

**12**

**1**

**DIAGRAMA DE CLASES**

Para el diseño de software, se elaboró un diagrama de clases Unified Modeling Language (UML), en el cual, proporcionará una explicación detallada de cada clase establecida en el diagrama, con sus respectivos, métodos, así como sus relaciones. Este sería el diagrama de clases:

Diagrama

Descripción generada automáticamente

Desglosando el diagrama:

* Existen 5 clases, las cuales serían cliente, producto, inventario, pago y pedido.

Cada clase esta compuesta de la siguiente manera:

* La clase ‘Cliente’ cuenta con todos sus atributos de tipo de protección (#), incluyendo el ‘id\_cliente’ de tipo entero, ‘nombre’, ‘apellido’, ‘num\_telefonico’, ‘direccion’, ‘email’ y ‘contraseña’ de tipo cadena de caracteres. Sus métodos serian 3 y todos de tipo de protección público (+), el método ‘creacionCuenta()’ retorna una cadena caracteres, y los métodos ‘realizarPedido()’ y ‘registrarCuenta()’ que son de tipo vacío.
* La clase ‘Producto’ se tienen atributos de visibilidad publica, los cuales son: ‘id\_producto’, y ‘stock’ de tipo entero, seguido de ‘nombre\_producto’, y ‘descripcion’ de tipo cadena de caracteres y ‘precio’ de tipo flotante. Los métodos que componen la clase son todos de visibilidad publica, ‘detallesProducto()’ retorna una cadena de caracteres, mientras que los métodos ‘restarStock()’, ‘agregarStock()’ y ‘actualizarPrecio()’ son de tipo vacio.

**13**

* La clase ‘Inventario’ este compuesto por un solo atributo de visibilidad privada (-), el cual es ‘productos’ que es un vector que contiene el objeto Producto. Y tiene 3 métodos, 2 de ellos de tipo vacío, los cuales serían: ‘agregarProducto()’ y ‘eliminarProducto()’, mientras que el método de ‘consultarStock()’ retorna un entero.
* La clase ‘Pago’ esta compuesto por atributos de visibilidad protegida, los cuales son: ‘id\_pago’ de tipo entero, ‘cliente’, ‘productos\_pedido’, ‘estado\_pedido’ y ‘metodo\_pago’ de tipo cadena de caracteres, teniendo de métodos ‘estadoPedido()’ y ‘totalPedido()’ de tipo vacio, ‘detallesPedido()’ que retorna una cadena de caracteres, y ‘fechaPedido()’ que retorna una variable de tipo fecha.
* Y, por último, la clase ‘Pedido’ esta compuesto por los atributos: ‘id\_pedido’ de tipo entero, ‘fecha\_pedido’ de tipo date, ‘estado’ de tipo cadena de caracteres, ‘total’ de tipo flotante y ‘productos’ un vector que contiene objetos de la clase Producto. Sus métodos de visibilidad publica y todos de tipo vacío: ‘agregarProductos()’, ‘actualizarPedido()’ y ‘actualizarEstado()’.

Una vez teniendo las clases definidas, las relaciones entre clases se definieron de la siguiente manera:

* La clase ‘Cliente’ está asociada con la clase ‘Pedido’.
* La clase ‘Pedido’ está asociada con la clase ‘Pago’ y la clase ‘Producto’.
* La clase ‘Inventario’ está asociada con la clase ‘Producto’.

Estas relaciones son bidireccionales, por lo que sus valores de multiplicidad serían los siguientes:

* Un cliente puede realizar varios pedidos, para cada pedido está asociada a un único cliente. Siendo una relación de 1 a muchos (1..).
* Un pedido puede contener múltiples productos, y un producto puede estar incluido en varios pedidos. Siendo una relación de muchos a muchos (n..m).
* Un inventario este compuesto por varios productos, pero cada producto pertenece a un único inventario. Siendo una relación de 1 a muchos (1..).
* Y, por último, cada pedido tiene asociada a un único pago, y cada pago corresponde a un único pedido. Siendo una relación de 1 a 1 (1..1).

**14**

**CLASIFICACIÓN DE PRODUCTOS DE CATÁLOGO**

Se les dio una clasificación a los productos pertenecientes al listado generado anteriormente, de modo que sea mucho más sencillo gestionar sus datos y disponibilidad, y poder implementarlos de forma más libre en una base de datos.

**Herramientas Manuales:**

* Destornillador plano, mediano – $50 MXN
* Destornillador Phillips, grande – $60 MXN
* Llave inglesa de 6" – $120 MXN
* Llave inglesa de 8" – $150 MXN
* Llave inglesa de 12" – $200 MXN
* Alicate universal, mediano – $120 MXN
* Alicate de corte, pequeño – $80 MXN
* Alicate de presión, grande – $250 MXN
* Martillo de carpintero, 400g – $90 MXN
* Martillo de goma, 300g – $70 MXN

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Medición:**

* Cinta métrica de 5m – $40 MXN
* Cinta métrica de 10m – $70 MXN
* Cinta métrica de 25m – $200 MXN
* Cinta métrica láser, 50m – $800 MXN

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Corte:**

* Serrucho para madera, 18" – $150 MXN
* Serrucho para metal, 12" – $120 MXN
* Cuchillo multiusos, retráctil – $50 MXN
* Cuchillo de hoja fija, 25mm – $80 MXN

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Jardinería:**

* Tijeras de podar, pequeñas – $150 MXN
* Tijeras de podar, grandes – $250 MXN

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**15**

**Accesorios de Taladro:**

* Brocas para madera, 3mm – $25 MXN
* Brocas para metal, 5mm – $30 MXN
* Brocas para concreto, 10mm – $40 MXN

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Fijación:**

* Clavos de acero, 1" (paquete de 100) – $30 MXN
* Tornillos para madera, 2" (paquete de 200) – $80 MXN

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Adhesivos:**

* Pegamento universal, 100g – $40 MXN
* Super glue, 20g – $25 MXN
* Silicona en tubo, 300ml – $100 MXN

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Material Eléctrico:**

* Bombilla LED 10W, blanca fría – $50 MXN
* Extensión eléctrica, 5m, 3 tomas – $150 MXN

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Accesorios Varios:**

* Cinta aislante negra, 19mm x 10m – $30 MXN
* Pintura en spray blanca, 400ml – $90 MXN
* Lija para madera, grano 80 – $40 MXN
* Lija para metal, grano 120 – $40 MXN

**16**

**DIAGRAMA DE ENTIDAD RELACIÓN**

Se definió el diagrama de entidad relación que tendrá la base de datos, identificando de qué forma interactuarán las tablas con base a sus contenidos.

Diagrama

Descripción generada automáticamente

**Componentes:**

* La entidad cliente, tiene como atributo el ID del cliente como llave primaria, nombre, apellido, correo electronico, direccion, contraseña y el numero telefonico como atributo multivaluado, ya que el cliente puede tener varios numeros telefonicos.
* La entidad pedido tiene como llave primaria el ID del pedido, seguido de los atributos monto total, fecha del pedido y el estado del pedido.
* La entidad pago tiene como llave priimaria el atributo el ID pago, seguido con el estado del pago, el metodo de pago y el monto total como atributos.
* La entidad carrito de compra tiene como llave primaria y atributo el ID del carrito de compra, el monto total y la cantidad productos que contiene el carrito.
* Y por ultimo, la entidad producto tiene el ID del producto como llave primaria y atributo, seguido del nombre del producto, precio, stock y la descripcion del producto.

**Relaciones:**

* La entidad cliente tiene una relacion de 1 a 1 con el carrito de compras, ya que un cliente tiene un carrito de compras, y un carrito de compras tiene un cliente.
* Tambien, la entidad cliente tiene otra relacion con la entidad de pedido, su relacion es de 1 a muchos, debido a que 1 o varios clientes pueden realizar un solo pedido, y un pedido es realizado por 1 o varios clientes.
* La entidad pedido tiene una relacion de 1 a 1 con la entidad carrito, porque un pedido puede cofnfirmar el carrito de compra y el carrito de compra puede confirmar el pedido.
* Seguido, el pedido tiene una relacion de igual forma de 1 a 1 con la entidad pago, porque un pedido confirma un pago, y un pago confirma un pedido.
* Y por ultimo, la entidad producto tiene una relacion de muchos a muchos con la entidad carrito, debido a que 1 o varios productos tienen 1 o varios carritos de compras, y a su vez, 1 o varios carritos de compras tienen 1 o mas productos.

**17**

**DIAGRAMA LÓGICO RELACIONAL**

Una vez teniendo el modelo entidad relación para nuestra base de datos, se realizó el modelo lógico relacional que el cual, explica cuál es la conexión que tienen las tablas usando llaves primarias y llaves foráneas, o también dicho de forma casual, 'Primary Key' y 'Foreign Key'. Para explicar la conexión de estas tablas, se debe considerar que para este tipo de diagramas se usa una notación especifica, las relaciones u conexiones que hay son las siguientes:

Tabla

Descripción generada automáticamente con confianza media

Las relaciones son las mismas, pero en este caso, se utilizarán las cardinalidades de ambos lados correspondientes de las entidades. Para determinar una relación, se tienen en cuenta estos dos lados. Con estos principios, se creó el diagrama lógico relacional de la siguiente manera:

Diagrama

Descripción generada automáticamente

**18**

**Diagrama desglosado:**

* La tabla cliente cuenta con una llave primaria denominada como “ClienteID”.
* La tabla carrito\_compras cuenta con una llave primaria denominada como “CarritoID”.
* La tabla pago cuenta con una llave primaria denominada como “PagoID”.
* La tabla producto cuenta con una llave primaria denominada como “ProductoID”.
* La tabla pedido cuenta con una llave primaria denominada como “PedidoID”.
* La tabla de carrito\_compra posee 2 llaves foráneas, las cuales serian “ClienteID” y “ProductoID”, están conectadas a sus respectivas tablas de donde provienen, Indicando que los productos están en el carrito y que el cliente le pertenece o tiene el carrito.
* La tabla pago tiene una llave foránea “PedidoID” conectada con la tabla de producto, indicando que el pedido esta vinculado con el pago.
* La tabla pedido tiene 2 llaves foráneas, las cuales son “ClienteID” y “CarritoID”, de igual forma conectadas a sus respectivas tablas. Esto quiere decir que el cliente realiza los pedidos y que el carrito pertenece a los pedidos.

**19**

**DISEÑO DE TABLAS**

Una vez definidos los diagrama, se diseñan los tipos de datos que tendría cada tabla para al momento que se implemente la base de datos, a continuación, estos serían los diseños de las tablas:

**Clientes:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo de dato** | **Tamaño** | **default** |
| ClienteID | int |  | NOT NULL |
| Nombre | varchar | 50 |  |
| Apellido | varchar | 50 |  |
| Num\_telefonico | varchar | 14 |  |
| Contraseña | varchar | 15 |  |
| Correo\_electronico | varchar | 100 |  |
| Direccion | varchar | 100 |  |

**Pago:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo de dato** | **Tamaño** | **default** |
| PagoID | int |  | NOT NULL |
| Estado\_pago | varchar | 15 |  |
| Método\_pago | varchar | 10 |  |
| Monto\_pago | decimal | 10,2 |  |

**Producto:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo de dato** | **Tamaño** | **Default** |
| ProductoID | int |  | NOT NULL |
| Precio | int |  |  |
| Descripcion | varchar | 200 |  |
| Nombre\_producto | varchar | 100 |  |
| Stock | Int |  |  |

**Pedido:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo de dato** | **Tamaño** | **Default** |
| PedidoID | int |  | NOT NULL |
| Monto\_total | decimal | 10,2 |  |
| Fecha\_pedido | date |  |  |
| Estado\_pedido | varchar | 15 |  |

**Carrito de compras:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo de dato** | **Tamaño** | **Default** |
| CarritoID | int |  | NOT NULL |
| Cantidad\_productos | int |  |  |
| Monto\_total | decimal | 10,2 |  |

**20**

**IMPLEMENTACIÓN DE BASE DE DATOS**

Utilizando todos los diagramas de la base de datos (entidad-relación, lógico relacional, diseño de tablas), se implementó en SQL una base de datos que respetara lo establecido por los diagramas, incluyendo las relaciones entre tablas, las llaves y todas las variables, y que permitiera el correcto funcionamiento al ser vinculado con la página de HTML, siendo el código de la base de datos el siguiente:

**21**

|  |
| --- |
| CREATE DATABASE Ferreteria;  USE Ferreteria;  CREATE TABLE Cliente (  ClienteID INT NOT NULL,  Nombre VARCHAR(50),  Apellido VARCHAR(50),  Num\_telefonico VARCHAR(14),  Contraseña VARCHAR(15),  Correo\_electronico VARCHAR(100),  Direccion VARCHAR(100),  PRIMARY KEY (ClienteID)  );  CREATE TABLE Producto (  ProductoID INT NOT NULL,  Precio INT,  Descripcion VARCHAR(200),  Nombre\_producto VARCHAR(100),  Stock INT,  PRIMARY KEY (ProductoID)  );  CREATE TABLE Carrito\_compras (  CarritoID INT NOT NULL,  Cantidad\_productos INT,  Monto\_total DECIMAL(10,2),  ClienteID INT NOT NULL,  ProductoID INT NOT NULL,  PRIMARY KEY (CarritoID),  FOREIGN KEY (ClienteID) REFERENCES Cliente(ClienteID),  FOREIGN KEY (ProductoID) REFERENCES Producto(ProductoID)  );  CREATE TABLE Pedido (  PedidoID INT NOT NULL,  Monto\_total DECIMAL(10,2),  Fecha\_pedido DATE,  Estado\_pedido VARCHAR(15),  ClienteID INT NOT NULL,  CarritoID INT NOT NULL,  PRIMARY KEY (PedidoID),  FOREIGN KEY (ClienteID) REFERENCES Cliente(ClienteID),  FOREIGN KEY (CarritoID) REFERENCES Carrito\_compras(CarritoID)  );  CREATE TABLE Pago (  PagoID INT NOT NULL,  Estado\_pago VARCHAR(15),  Metodo\_pago VARCHAR(10),  Monto\_pago DECIMAL(10,2),  PedidoID INT NOT NULL,  PRIMARY KEY (PagoID),  FOREIGN KEY (PedidoID) REFERENCES Pedido(PedidoID)  ); |

**22**

**IMPLEMENTACIÓN DE INTERFAZ**

Haciendo uso de todos los documentos anteriores, de implementó el diseño de Vista del sistema a través de HTML, CSS y JS, de modo que se pudiera cumplir con todos los requisitos establecidos.

Los códigos más importantes de la vista del sistema son los siguientes:

* Index.
* Main.
* Login.

**Index:**

El archivo index.html es el archivo principal que muestra la página inicial del e-business, la cual está conectada a todos los demás archivos de funcionamiento:

**1**

**23**

|  |
| --- |
| <!DOCTYPE html>  <html lang="es">  <head>      <meta charset="UTF-8">      <meta http-equiv="X-UA-Compatible" content="IE=edge">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Ferretería E-business</title>      <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.9.1/font/bootstrap-icons.css">      <link rel="stylesheet" type="text/css" href="https://cdn.jsdelivr.net/npm/toastify-js/src/toastify.min.css">      <link rel="stylesheet" href="./css/main.css">  </head>  <body>      <div class="wrapper">          <header class="header-mobile">              <h1 class="logo">Ferretería E-business</h1>            </header>          <aside>              <button class="close-menu" id="close-menu">                  <i class="bi bi-x"></i>              </button>              <header>                  <h1 class="logo">Ferretería E-business</h1>              </header>              <nav>                  <ul class="menu">                      <li>                          <button id="todos" class="boton-menu boton-categoria active"><i class="bi bi-hand-index-thumb-fill"></i> Todos los productos</button>                      </li>                      <li>                          <button id="manuales" class="boton-menu boton-categoria"><i class="bi bi-tools"></i> Manuales</button>                      </li>                      <li>                          <button id="medicion" class="boton-menu boton-categoria"><i class="bi bi-bricks"></i> Medición</button>                      </li>                      <li>                          <button id="corte" class="boton-menu boton-categoria"><i class="bi bi-archive"></i> Corte</button>                      </li>                      <li>                          <button id="jardineria" class="boton-menu boton-categoria"><i class="bi bi-scissors"></i> Jardinería</button>                      </li>                      <li>                          <button id="taladro" class="boton-menu boton-categoria"><i class="bi bi-hammer"></i> Accesorios de taladro</button>                      </li>                      <li>                          <button id="fijacion" class="boton-menu boton-categoria"><i class="bi bi-asterisk"></i> Fijación</button>                      </li>                      <li>                          <button id="adhesivos" class="boton-menu boton-categoria"><i class="bi bi-droplet"></i> Adhesivos</button>                      </li>                      <li>                          <button id="electrico" class="boton-menu boton-categoria"><i class="bi bi-lightning-fill"></i> Material eléctrico</button>                      </li>                      <li>                          <button id="varios" class="boton-menu boton-categoria"><i class="bi bi-check-circle"></i> Accesorios varios</button>                      </li>                      <li>                          <a class="boton-menu boton-carrito" href="./carrito.html">                              <i class="bi bi-cart-fill"></i> Carrito <span id="numerito" class="numerito">0</span>                          </a>                          </a>                          <a class="log-sign-in" href="./login.html">                              <i class="bi bi-person-fill"></i> Iniciar sesión  **24**                          </a>                      </li>                  </ul>              </nav>              <footer>                  <p class="texto-footer">© 2024 Ferretería E-business</p>              </footer>          </aside>          <main>              <h2 class="titulo-principal" id="titulo-principal">Todos los productos</h2>              <div id="contenedor-productos" class="contenedor-productos">              </div>          </main>      </div>      <div id="loggedInMessage" class="logged-in" style="display: none;">          <p>¡Bienvenido, cliente!</p>      </div>      <script>          function checkLoginStatus() {              const user = localStorage.getItem('user');              if (user) {                  document.getElementById('loggedInMessage').style.display = 'block';              }          }          checkLoginStatus();      </script>      <script type="text/javascript" src="https://cdn.jsdelivr.net/npm/toastify-js"></script>      <script src="./js/main.js"></script>      <script src="./js/menu.js"></script>  </body>  </html> |

**Login:**

Cumple con la función de ser el requisito de permitir al usuario crear usuarios e iniciar sesión, para la gestión de cuentas:

**25**

|  |
| --- |
| <!DOCTYPE html>  <html lang="es">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Index - Ferretería E-business</title>      <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-icons.css">      <link rel="stylesheet" href="./css/login.css">      <style>          body {              font-family: 'Rubik', sans-serif;              background-color: #f5f5f5;              margin: 0;              padding: 0;              display: flex;              align-items: center;              justify-content: center;              min-height: 100vh;          }          .container {              width: 100%;              max-width: 400px;              background-color: #ffffff;              padding: 2rem;              border-radius: 8px;              box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);          }          h2 {              color: var(--clr-main, #4b33a8);              text-align: center;              margin-bottom: 1.5rem;          }          .form-group {              margin-bottom: 1rem;          }          label {              display: block;              margin-bottom: 0.5rem;              color: #333333;          }          input[type="email"],          input[type="password"],          input[type="text"],          input[type="tel"],          textarea {              width: 100%;              padding: 0.75rem;              border: 1px solid #cccccc;              border-radius: 4px;              font-size: 1rem;          }          button {              width: 100%;              padding: 0.75rem;  **26**              background-color: var(--clr-main, #4b33a8);              color: #ffffff;              border: none;              border-radius: 4px;              font-size: 1rem;              cursor: pointer;              margin-top: 1rem;          }          button:hover {              background-color: var(--clr-main-light, #785ce9);          }          .actions {              display: flex;              justify-content: space-between;              margin-top: 1rem;          }          .actions a {              color: var(--clr-main, #4b33a8);              text-decoration: none;          }          .actions a:hover {              text-decoration: underline;          }          .logged-in {              text-align: center;              margin-top: 2rem;          }          .logged-in button {              background-color: #d9534f;              color: white;          }      </style>  </head>  <body>      <div class="container">          <h2>Iniciar Sesión</h2>          <form id="loginForm">              <div class="form-group">                  <label for="email">Correo Electrónico</label>                  <input type="email" id="email" name="email" required placeholder="Ingresa tu correo">              </div>              <div class="form-group">                  <label for="password">Contraseña</label>  **27**                  <input type="password" id="password" name="password" required placeholder="Ingresa tu contraseña">              </div>              <button type="submit">Iniciar Sesión</button>          </form>          <div class="actions">              <a href="./recuperar-contrasena.html">¿Olvidaste tu contraseña?</a>              <a href="#crear-cuenta">Crear cuenta</a>          </div>          <h2 id="crear-cuenta" style="margin-top: 2rem;">Crear Cuenta</h2>          <form id="registerForm">              <div class="form-group">                  <label for="nombre">Nombre</label>                  <input type="text" id="nombre" name="nombre" required placeholder="Ingresa tu nombre">              </div>              <div class="form-group">                  <label for="apellido">Apellido</label>                  <input type="text" id="apellido" name="apellido" required placeholder="Ingresa tu apellido">              </div>              <div class="form-group">                  <label for="telefono">Teléfono</label>                  <input type="tel" id="telefono" name="telefono" required placeholder="Ingresa tu número de teléfono">              </div>              <div class="form-group">                  <label for="email-registro">Correo Electrónico</label>                  <input type="email" id="email-registro" name="email" required placeholder="Ingresa tu correo">              </div>              <div class="form-group">                  <label for="password-registro">Contraseña</label>                  <input type="password" id="password-registro" name="password" required placeholder="Crea una contraseña">              </div>              <div class="form-group">                  <label for="direccion">Dirección</label>                  <textarea id="direccion" name="direccion" required placeholder="Ingresa tu dirección"></textarea>              </div>              <button type="submit">Crear Cuenta</button>          </form>      </div>      <div id="loggedInMessage" class="logged-in" style="display: none;">          <p>¡Bienvenido, cliente!</p>          <button id="logoutBtn">Cerrar Sesión</button>      </div>  **28**      <script>          // Función para mostrar mensaje de bienvenida si la sesión está activa          function checkLoginStatus() {              const user = localStorage.getItem('user');              if (user) {                  document.getElementById('loginForm').style.display = 'none';                  document.getElementById('registerForm').style.display = 'none';                  document.getElementById('loggedInMessage').style.display = 'block';              } else {                  document.getElementById('loginForm').style.display = 'block';                  document.getElementById('registerForm').style.display = 'block';                  document.getElementById('loggedInMessage').style.display = 'none';              }          }          // Llamar a la función para verificar el estado de la sesión al cargar la página          checkLoginStatus();          // Iniciar sesión (simulado)          document.getElementById('loginForm').addEventListener('submit', function (e) {              e.preventDefault();              localStorage.setItem('user', 'cliente'); // Guardamos el nombre de usuario como "cliente"              window.location.href = "index.html"; // Redirige al index.html          });          // Crear cuenta (simulado)          document.getElementById('registerForm').addEventListener('submit', function (e) {              e.preventDefault();              localStorage.setItem('user', 'cliente'); // Guardamos el nombre de usuario como "cliente"              window.location.href = "index.html"; // Redirige al index.html          });          // Cerrar sesión          document.getElementById('logoutBtn').addEventListener('click', function () {              localStorage.removeItem('user'); // Eliminar el usuario del localStorage              window.location.href = "index.html"; // Redirige al index.html para que se actualice la vista          });      </script>  </body>  </html> |

**Main:**

Utiliza todos los datos proporcionados por otros archivos como productos.json para poder generar los recuadros y gestionar las ventas del usuario.

**29**

|  |
| --- |
| let productos = [];  fetch(".Dia4/js/productos.json")      .then(response => response.json())      .then(data => {          productos = data;          cargarProductos(productos);      })  const contenedorProductos = document.querySelector("#contenedor-productos");  const botonesCategorias = document.querySelectorAll(".boton-categoria");  const tituloPrincipal = document.querySelector("#titulo-principal");  let botonesAgregar = document.querySelectorAll(".producto-agregar");  const numerito = document.querySelector("#numerito");  botonesCategorias.forEach(boton => boton.addEventListener("click", () => {      aside.classList.remove("aside-visible");  }))  function cargarProductos(productosElegidos) {      contenedorProductos.innerHTML = "";      productosElegidos.forEach(producto => {          const div = document.createElement("div");          div.classList.add("producto");          div.innerHTML = `              <img class="producto-imagen" src="${producto.imagen}" alt="${producto.titulo}">              <div class="producto-detalles">                  <h3 class="producto-titulo">${producto.titulo}</h3>                  <p class="producto-precio">$${producto.precio}</p>                  <button class="producto-agregar" id="${producto.id}">Agregar</button>              </div>          `;          contenedorProductos.append(div);      })      actualizarBotonesAgregar();  }  botonesCategorias.forEach(boton => {      boton.addEventListener("click", (e) => {          botonesCategorias.forEach(boton => boton.classList.remove("active"));          e.currentTarget.classList.add("active");          if (e.currentTarget.id != "todos") {              const productoCategoria = productos.find(producto => producto.categoria.id === e.currentTarget.id);  **30**              tituloPrincipal.innerText = productoCategoria.categoria.nombre;              const productosBoton = productos.filter(producto => producto.categoria.id === e.currentTarget.id);              cargarProductos(productosBoton);          } else {              tituloPrincipal.innerText = "Todos los productos";              cargarProductos(productos);          }      })  });  function actualizarBotonesAgregar() {      botonesAgregar = document.querySelectorAll(".producto-agregar");      botonesAgregar.forEach(boton => {          boton.addEventListener("click", agregarAlCarrito);      });  }  let productosEnCarrito;  let productosEnCarritoLS = localStorage.getItem("productos-en-carrito");  if (productosEnCarritoLS) {      productosEnCarrito = JSON.parse(productosEnCarritoLS);      actualizarNumerito();  } else {      productosEnCarrito = [];  }  function agregarAlCarrito(e) {      Toastify({          text: "Producto agregado",          duration: 3000,          close: true,          gravity: "top", // `top` or `bottom`          position: "right", // `left`, `center` or `right`          stopOnFocus: true, // Prevents dismissing of toast on hover          style: {            background: "linear-gradient(to right, #4b33a8, #785ce9)",            borderRadius: "2rem",            textTransform: "uppercase",            fontSize: ".75rem"          },          offset: {              x: '1.5rem', // horizontal axis - can be a number or a string indicating unity. eg: '2em'              y: '1.5rem' // vertical axis - can be a number or a string indicating unity. eg: '2em'            },          onClick: function(){} // Callback after click  **31**        }).showToast();      const idBoton = e.currentTarget.id;      const productoAgregado = productos.find(producto => producto.id === idBoton);      if(productosEnCarrito.some(producto => producto.id === idBoton)) {          const index = productosEnCarrito.findIndex(producto => producto.id === idBoton);          productosEnCarrito[index].cantidad++;      } else {          productoAgregado.cantidad = 1;          productosEnCarrito.push(productoAgregado);      }      actualizarNumerito();      localStorage.setItem("productos-en-carrito", JSON.stringify(productosEnCarrito));  }  function actualizarNumerito() {      let nuevoNumerito = productosEnCarrito.reduce((acc, producto) => acc + producto.cantidad, 0);      numerito.innerText = nuevoNumerito;  }  const historialVentas = [      { id: 1, fecha: "2024-11-20", producto: "Martillo", cantidad: 2, total: "$40" },      { id: 2, fecha: "2024-11-21", producto: "Destornillador", cantidad: 1, total: "$15" },      { id: 3, fecha: "2024-11-21", producto: "Taladro", cantidad: 1, total: "$120" }  ];  function cargarHistorialVentas() {      const tablaCuerpo = document.querySelector("#tabla-ventas tbody");      historialVentas.forEach(venta => {          const fila = document.createElement("tr");          fila.innerHTML = `              <td style="border: 1px solid #ddd; padding: 8px;">${venta.id}</td>              <td style="border: 1px solid #ddd; padding: 8px;">${venta.fecha}</td>              <td style="border: 1px solid #ddd; padding: 8px;">${venta.producto}</td>              <td style="border: 1px solid #ddd; padding: 8px;">${venta.cantidad}</td>              <td style="border: 1px solid #ddd; padding: 8px;">${venta.total}</td>          `;          tablaCuerpo.appendChild(fila);      });  }  document.addEventListener("DOMContentLoaded", cargarHistorialVentas); |

**32**

**PRUEBAS UNITARIAS**

Para poder probar que la autenticación de usuarios se realizaron inicios de sesión constantes, de modo que se pudiera observar si algunos de los siguientes casos se presentaba:

* Error en la detección de cuenta.
* Que indique que la cuenta que no existe.
* Ingresar a una cuenta equivocada.

Afortunadamente, ninguno de esos errores se presentó, y el inicio de sesión por su cuenta funciona correctamente.

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

Se probó si el carrito de compras funcionaba en su redireccionamiento de página, así como en la muestra de productos cuando se selecciona alguno, así como indicar que no hay productos en caso de no seleccionar nada, contando con ningún problema al respecto.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

De igual forma, se calculan los totales de precios de manera adecuada, realizando las sumas indicadas.

**33**

**PRUEBAS DE INTEGRACIÓN**

Para comprobar que el catálogo de productos funciona y se visualiza correctamente en la página, debe de tener una interacción correcta con archivos como “main.js”, ya que administra todos los productos que posee el archivo “productos.json” de modo que permite la selección de los mismos, así como su compra.

Inicialmente, se presentaron problemas para que los productos fueran mostrados en la página, esto se debió a que la redirección de estos era la incorrecta.

**PRUEBAS DE FUNCIONAMIENTO**

Para comprobar que las notificaciones funcionaban correctamente, se realizan actualizaciones durante la utilización del sistema por parte de los desarrolladores, de modo que notificara al usuario en el momento.

**PRUEBAS DE INTERFAZ DE USUARIO**

Para poder poner a prueba el funcionamiento de la interfaz de usuario, los desarrolladores realizan todas las posibles acciones dentro del sistema, donde se incluyen los siguientes casos:

* Inicio de sesión.
* Registrar usuario.
* Consultar productos por categorías.
* Consultar carrito.
* Agregar productos al carrito.
* Comprar productos.

**PRUEBAS DE RENDIMIENTO**

Se utiliza la página de manera normal durante un tiempo determinado por los desarrolladores para la detección de errores y revisión de rendimiento, donde se evalúa el tiempo de respuesta del sistema a una acción solicitada por el usuario, de modo que todos estaban utilizando el sistema al mismo tiempo.

Al igual que en la prueba de funcionamiento de notificaciones, esto permitiría observar el comportamiento de las notificaciones mientras el sistema es utilizado por muchos usuarios de manera simultánea, evaluando la efectividad de esta.

**34**

**MANTENIMIENTO CORRECTIVO**

A lo largo del proceso de desarrollo del sistema, el equipo de desarrollo puede presentar errores o problemas que pueden afectar de manera leve o grave a futuro, por lo cual se deben tomar medidas que busquen eliminar cualquier clase de problema que se pueda presentar después del producto final, algunos de estos posibles problemas son:

* Problemas en el código.
* Bajo rendimiento del sistema.
* Fallos o crasheos.
* Error al mostrar datos o archivos en la interfaz.
* Problemas de redireccionamiento.
* Error en la base de datos.

Cada problemática cuenta con su respectiva solución que se aplica según el contexto y la gravedad de esta.

La mejor forma de detectar aquellos errores que presentan molestias para el usuario es a través de la retroalimentación que brindan por medio de reseñas y opiniones, lo que hace esencial programar una revisión de reseñas de usuario de cada 7 o 15 días, dependiendo de la gravedad de la falla.

**MANTENIMIENTO PREVENTIVO**

Haciendo uso de las pruebas de funcionamiento, implementación y de unidad que intentan comprobar si los requisitos se cumplen en diferentes contextos, se realizan mantenimientos preventivos a cada requisito funcional y no funcional del sistema de manera periódica, estableciendo fechas en las cuales se realizan dichas pruebas para la detección temprana de errores, estas revisiones pueden incluir:

* Revisión de código.
* Revisión de botones.
* Revisión de base de datos.

En caso de detectar algún problema temprano, se toman las medidas necesarias según corresponda.

Así mismo, el mantenimiento preventivo programado busca detectar si alguna de las dependencias del sistema se encuentra obsoleta, para poder realizar los cambios y medidas de seguridad indicados.

El periodo ideal para la realización de mantenimiento preventivo es cada 7 días, ya que de ese modo puede coincidir con las revisiones de reseñas de usuarios para el mantenimiento correctivo, lo que permite una mejor visión de la problemática o posible problemática.

**35**

**MANTENIMIENTO PERFECTIVO**

Para la integración de mantenimiento perfectivo, se realizan revisiones periódicas de las opiniones y comentarios dejados por usuarios con respecto a su experiencia utilizando el sistema, y analizando los resultados se obtienen aquellos considerados de suma importancia, como aquellos que se hayan mencionado de manera muy frecuente por los usuarios, demostrando así una demanda por parte de los consumidores ante el funcionamiento de una característica del sistema o el propio sistema.

Lo ideal es realizar las recopilaciones de datos cada 30 días para dar margen al mantenimiento preventivo y correctivo, así como la creación de un plan de desarrollo que permita implementar todos los nuevos requisitos que presenten los usuarios durante la retroalimentación.

Una vez obtenidos los resultados de la retroalimentación de usuarios, se toman medidas para poder realizar las mejoras o cambios solicitados para brindar una mejor experiencia de usuario, donde se encuentran cambios y mejoras que van desde lo estético hasta lo funcional.

**MANTENIMIENTO ADAPTATIVO**

La asignación de un equipo de estudio y monitoreo de tecnologías es importante, debido a que permite identificar de qué forma se puede adaptar el sistema a las nuevas tecnologías, brindando una mayor ventaja competitiva al integrar nuevas versiones de software, y establecer una nueva estructura de manera secuencial en el sistema, preparándolo para los futuros cambios a nivel de diseño, así como requisitos legales, etc.

La mejor forma de realizar dichos mantenimientos adaptativos es durante la realización de mantenimientos perfectivos, ya que brindan facilidades al estar brindando mejoras de calidad y diseño al sistema, por lo que lo ideal es realizarse cada 30 día

**36**