7/3/25

Proyecto Final

Sistema de Seguimiento de Pedidos

Daniela Caceres Sanchez

(Proyecto Final): "Sistema de Seguimiento de Pedidos"

Introducción: El sistema de seguimiento de pedidos sirve para gestionar de manera eficiente los pedidos de los clientes. A través de un diseño estructurado con clases, listas enlazadas y algoritmos de búsqueda y ordenamiento, se logra un manejo dinámico de los pedidos, asegurando su correcta administración.

El objetivo principal es la solución sencilla pero funcional que permita registrar, buscar, ordenar y eliminar pedidos de manera eficiente.

Este proyecto permite realizar pedidos de clientes de manera sencilla, implementa clases para modelar productos, consumidores y pedidos, utiliza lista enlazada para la administración de secuencia de pedidos.

El sistema permite

- El registro de nuevos pedidos.
- Buscar pedidos por ID (búsqueda lineal)
- Ordenar pedidos según criterios como fecha o prioridad.
- Marcar pedidos como procesados mediante arreglos binario.
- Eliminar pedidos o productos específicos dentro de un pedido.

Diseño del Código

1. Modelado de Entidades con Clases:
   Se definen las clases en JavaScript para representar los elementos

- Productos: Representa los productos con un id, nombre y precio.

```
// Clase Productos
class Productos {
    constructor(id, nombre, precio) {
        if (!id || !nombre || !precio) {
            throw new Error("Todos los campos son obligatorios");
        }
        this.id = id;
        this.nombre = nombre;
        this.precio = precio;
    }
}
```

- Consumidor: Representa a los clientes con un id y nombre.

```

// Clase Consumidor
class Consumidor {
    constructor(id, nombre) {
        if (!id || !nombre) {
            throw new Error("Todos los campos son obligatorios");
        }
```

```
        this.id = id;
        this.nombre = nombre;
    }
}
```

- Sistema De Pedido: Define los pedidos, vinculando un consumidor con una lista de productos, además de almacenar la fecha, prioridad y estado de procesamiento.

```
// Clase SistemaDePedido
class SistemaDePedido {
    constructor(id, consumidor, productos, fecha, prioridad) {
        this.id = id;
        this.consumidor = consumidor;
        this.productos = productos;
        this.fecha = fecha;
        this.prioridad = prioridad;
        this.procesado = false; // Estado en binario (false = no procesado, true = procesado)
    }

    marcarProcesado() {
        this.procesado = true;
    }

    eliminarProducto(productoId) {
        const index = this.productos.findIndex(producto => producto.id === productoId);
        if (index === -1) {
            console.log(`Producto con ID ${productoId} no encontrado`);
            return;
        }
        this.productos.splice(index, 1);
    }
}
```

2. Gestión Dinámica de Pedidos con Lista Enlazada

   Para manejar los pedidos de manera eficiente, se implementó una lista enlazada en la clase Linkedlist.

   Cada pedido se almacena en un Nodo, que contiene una referencia al siguiente pedido, formando una cadena de pedido enlazados.

   - Agrego el Nodo.

```

// Clase Nodo para Lista Enlazada
class Nodo {
```

```
    constructor(pedido) {
        this.pedido = pedido;
        this.siguiente = null;
    }
}
```

- Agrega pedidos al final de la lista.
- Eliminar pedidos por ID.
- Buscar un pedido especifico mediante búsqueda lineal.
- Ordenar pedidos según diferentes criterios (fecha o prioridad).

```
// Clase LinkedList (Lista Enlazada)
class LinkedList {
    constructor() {
        this.cabeza = null;
    }

    aggFinal(pedido) {
        const nuevoNodo = new Nodo(pedido);
        if (!this.cabeza) {
            this.cabeza = nuevoNodo;
        } else {
            let actual = this.cabeza;
            while (actual.siguiente) {
                actual = actual.siguiente;
            }
            actual.siguiente = nuevoNodo;
        }
    }

    eliminarPedidoPorId(id) {
        if (!this.cabeza) {
            console.log("La lista está vacía");
            return;
        }

        if (this.cabeza.pedido.id === id) {
            this.cabeza = this.cabeza.siguiente;
            console.log(`Pedido con ID ${id} eliminado`);
            return;
        }

        let actual = this.cabeza;
        while (actual.siguiente && actual.siguiente.pedido.id !== id)
{
            actual = actual.siguiente;
```

```
        }

        if (!actual.siguiente) {
            console.log(`Pedido con ID ${id} no encontrado`);
            return;
        }

        actual.siguiente = actual.siguiente.siguiente;
        console.log(`Pedido con ID ${id} eliminado`);
    }

    buscarPedido(id) {
        let actual = this.cabeza;
        while (actual) {
            if (actual.pedido.id === id) {
                return actual.pedido;
            }
            actual = actual.siguiente;
        }
        return null;
    }

    ordenarCriterio(criterio) {
        if (!this.cabeza || !this.cabeza.siguiente) {
            return;
        }

        let listaOrdenada = null;
        let actual = this.cabeza;

        while (actual) {
            const siguiente = actual.siguiente;

            if (!listaOrdenada || actual.pedido[criterio] <
listaOrdenada.pedido[criterio]) {
                actual.siguiente = listaOrdenada;
                listaOrdenada = actual;
            } else {
                let temp = listaOrdenada;
                while (temp.siguiente &&
temp.siguiente.pedido[criterio] < actual.pedido[criterio]) {
                    temp = temp.siguiente;
                }
                actual.siguiente = temp.siguiente;
                temp.siguiente = actual;
```

```
            }
            actual = siguiente;
        }
        this.cabeza = listaOrdenada;
    }

    mostrarLista() {
        let actual = this.cabeza;
        while (actual) {
            console.log(actual.pedido);
            actual = actual.siguiente;
        }
    }
}
```

3. Administración de Pedidos

   La clase Pedido Administrado actúa como un controlador del sistema, el cual centraliza las operaciones sobre la estructura de datos, el cual se encarga de:

- Registrar nuevos pedidos como un identificar único.
- Ordenar pedidos según criterios definidos.
- Buscar pedidos por ID dentro de la lista enlazada.
- Mostrar la lista completa de pedidos.

```
// Clase PedidoAdministrado
class PedidoAdministrado {
    constructor() {
        this.pedidos = new LinkedList();
        this.contadorId = 1;
    }

    agregarPedido(consumidor, productos, fecha, prioridad) {
        const nuevoPedido = new SistemaDePedido(this.contadorId++,
consumidor, productos, fecha, prioridad);
        this.pedidos.aggFinal(nuevoPedido);
        return nuevoPedido;
    }

    buscarPedidoPorId(id) {
        return this.pedidos.buscarPedido(id);
    }

    eliminarPedido(id) {
        this.pedidos.eliminarPedidoPorId(id);
    }

```

```
    ordenarPedidos(criterio) {
        this.pedidos.ordenarCriterio(criterio);
    }

    mostrarPedidos() {
        this.pedidos.mostrarLista();
    }
}
```

4. Implementación de Algoritmos
   El código incorpora diferentes algoritmos para mejorar la funcionalidad del sistema:

- Búsqueda Lineal: Se utiliza para encontrar pedidos dentro de la lista enlazada.
- Ordenamiento por Criterio: Se implementa un algoritmo de inserción para ordenar los pedidos por fecha o prioridad.
- Arreglo Binario para Estado: Cada pedido tiene un estado de procesamiento (true o false), simulando un arreglo binario para indicar si ha sido procesado o no.

5. Creación de Ejemplos y Pruebas

   Al final del código, se crean instancias de Productos y Consumidor, y se generan varios pedidos con diferentes prioridades. Luego, se muestra la lista de pedidos registrados. Esto permite probar la funcionalidad del sistema y validar su correcto funcionamiento.

```
//creo un administrador de pedidos
const administrador = new PedidoAdministrado();

// Crear un consumidor
const consumidor1 = new Consumidor(301, "Carlos Gómez");

// Crear productos
const producto1 = new Productos(101, "Teclado", 50);
const producto2 = new Productos(102, "Monitor", 20);
const productos = [producto1, producto2];

// Agregar pedidos con diferentes prioridades
const pedidoAlto = administrador.agregarPedido(consumidor1, productos,
"2025-03-07", "Alta");
const pedidoMedio = administrador.agregarPedido(consumidor1,
[producto1], "2025-03-08", "Media");
const pedidoBajo = administrador.agregarPedido(consumidor1,
[producto2], "2025-03-09", "Baja");

// Mostrar todos los pedidos
console.log(" Lista de pedidos:");
```

```
administrador.mostrarPedidos();

// Buscar un pedido específico
console.log("🔍 Buscando pedido con ID:", pedidoAlto.id);
console.log(administrador.buscarPedidoPorId(pedidoAlto.id));

// Eliminar un pedido
console.log(" Eliminando pedido...");
administrador.eliminarPedido(pedidoAlto.id);
administrador.mostrarPedidos();

// Ordenar pedidos por prioridad
console.log(" Ordenando pedidos por prioridad:");
administrador.ordenarPedidos("prioridad");
administrador.mostrarPedidos();
```