Asignatura: Estructuras de Datos

LEONARDO NARVAEZ, MARISCAL MESIAS, MURILLO GABRIEL, REA DENISE

Departamento de Ciencias De La Computación,

Universidad de Las Fuerzas Armadas ESPE.

Ingeniería en Desarrollo de Software, NRC 14544,

Sangolquí, Ecuador.

PROYECTO PRIMER PARCIAL:

MANUAL TÉCNICO:

SISTEMA PARA COMPRAS DE ALIMENTOS "ESPESPRESS"

LÓGICA DE FLUJO DEL SISTEMA

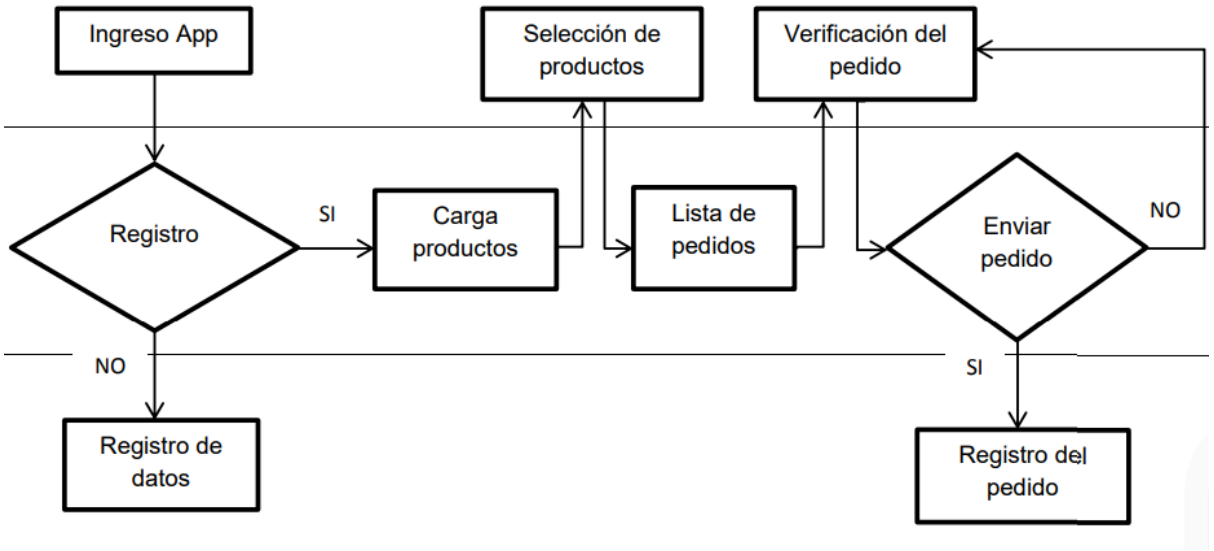


Figura 1. Flujograma del Sistema. (Tomado de: Borbor Villón, M. C. 2019, p. 21)

ESTRUCTURA PRINCIPAL

A continuación tenemos una descripción de la estructura central del proyecto, que es una lista doblemente enlazada. Esto lo podemos observar en la estructura *"Nodo"*, ya que, tiene como atributos un puntero al Nodo llamado *"siguiente"* y otro puntero al Nodo *"anterior"*.

```
template<typename T>
struct Nodo {
    int contador;
    T dato;
    Nodo* siguiente;
    Nodo* anterior;
};
```

En cambio, la lista tiene un puntero de tipo genérico cualquiera llamado "cabeza", que en este caso representa el primer elemento de la lista.

```
template <typename T>
class Lista {
public:
    Nodo<T>* cabeza;

public:
    Lista() : cabeza(nullptr) {}
    Nodo<T>* getCabeza(){
        return cabeza;
    }
```

Destructor de la Lista:

```
~Lista(){
    Nodo<T>* actual = cabeza;
    while (actual != nullptr) {
        Nodo<T>* siguiente = actual->siguiente;
        delete actual;
        actual = siguiente;
    }
    cabeza = nullptr;
}
```

Enseguida tenemos las funciones que están declaradas en la clase Lista y que son con las que vamos a recorrer, buscar, eliminar, etc. implementando esta lista para hacer una lista de estudiante, de productos, y de pedidos.

```
void insertarAlFinal(T valor) {
    Nodo<T>* nuevo = new Nodo<T>;
    nuevo->dato = valor;
    nuevo->siguiente = nullptr;
    nuevo->anterior = nullptr;
    nuevo->contador = 1;

    if (!cabeza) {
        cabeza = nuevo;
    } else {
        Nodo<T>* actual = cabeza;
        while (actual->siguiente) {
            actual = actual->siguiente;
        }
        actual->siguiente = nuevo;
        nuevo->anterior = actual;
    }
}
```

La función *"insertarComida"* tiene como objetivo principal insertar un producto en la lista y, si el producto ya existe en la lista, incrementar su contador.

```
void insertarComida(T valor){
    Nodo<T>* actual = cabeza;
    bool repetido = false;
    while(actual != nullptr){
        if(actual->dato == valor){
            actual->contador++;
            repetido = true;
            break;
        }
        actual = actual->siguiente;
    }
    if(!repetido) {
        Nodo<T>* nuevo = new Nodo<T>;
        nuevo->dato = valor;
        nuevo->siguiente = nullptr;
        nuevo->anterior = nullptr;
        nuevo->contador = 1;

        if (!cabeza) {
            cabeza = nuevo;
        } else {
            Nodo<T>* actual = cabeza;
            while (actual->siguiente) {
                actual = actual->siguiente;
            }
            actual->siguiente = nuevo;
            nuevo->anterior = actual;
        }
    }
}
```

Imprimir Datos de la lista:

```
void imprimirLista()const{
    Nodo<T>* actual = cabeza;
    while (actual) {
        actual->dato.mostrarDatos();
        //cout<<actual->dato<<endl;
        actual = actual->siguiente;
    }
    cout << endl;
}
```

La siguiente función imprime una lista pero con todos los productos que existen en el bar de la Universidad de Las Fuerzas Armadas.

```
string imprimirListaComida(float& pago) const {
    Nodo<T>* actual = cabeza;
    string total;
    pago=0;
    while (actual) {
        float precioProducto=actual->dato.getPrecio();
        int cantidad=actual->contador;
        float precio=precioProducto*cantidad;
        cout << "Producto: "+actual->dato.getDescripcion()<<endl;
        cout << "Precio:                          $"+to_string(precioProducto*cantidad)<<endl;
        cout << "Cantidad: "+to_string(cantidad)<<endl;
        pago+=precio;
        actual = actual->siguiente;
    }
    total+="Precio total de la orden:             $"+to_string(pago);
    return total;
}
```

Por último tenemos la función que imprime el menú de productos del bar.

```
void imprimirMenu(){
    Nodo<T>* actual=cabeza;
    int cont=1;
    while(actual){
        cout<<cont<<". "<<actual->dato.getDescripcion()<<"........$ "<<actual->dato.getPrecio()<<endl;
        actual=actual->siguiente;
        cont++;
    }
}
```

Función para eliminar un nodo:

```
bool eliminar(T& valor){
    Nodo<T>* actual=cabeza;
    Nodo<T>* anterior=NULL;
    while(actual != NULL && actual -> dato != valor){
        anterior = actual;
        actual = actual->siguiente;
    }
    if(actual==NULL){
        cout<<"El producto ingresado no existe en la carta."<<endl;
        return false;
    }
    if(actual->contador > 1){
            actual->contador--;
    }else {
        Nodo<T>* siguiente = actual->siguiente;
        if (anterior != nullptr) {
            anterior->siguiente = siguiente;
        } else {
            cabeza = siguiente;
        }
        if (siguiente != nullptr) {
            siguiente->anterior = anterior;
        }
        delete actual;
        cout << "Se ha eliminado " << valor.getDescripcion() << endl;
    }
    return true;
}
```

Función para buscar en la lista:

```
bool buscar(T valor) {
    if (esVacia()) {
        cout << "La lista está vacía, no se puede buscar" << endl
        return false;
    }

    Nodo<T>* tmp = cabeza;
    while (tmp) {
        if (tmp->dato == valor) {
            return true;
        }
        tmp = tmp->siguiente;
    }

    cout << "El número no está en la lista" << endl;
    return false;
}

bool esVacia() {
    return cabeza == nullptr;
}
```

ROLES DE USUARIO

El presente sistema está pensado para 2 tipos de usuarios o roles, donde cada uno interactúa con el sistema y tienen sus comportamientos propios, definidos en los casos de uso que se describen a continuación.

Tenemos 2 usuarios: El estudiante y el Administrador del bar:

Para el Estudiante:

Registrar Cuenta: Permite a un nuevo estudiante crear una cuenta en el sistema.

Iniciar sesión: Permite a un estudiante registrado acceder a su cuenta.

Ver Productos: Permite a un estudiante visualizar los productos disponibles para compra.

Crear Pedido: Permite a un estudiante seleccionar y agregar productos a un nuevo pedido.

Ver Historial de Pedidos: Permite a un estudiante ver todos sus pedidos anteriores.

Pagar Pedido: Permite a un estudiante completar el pago de un pedido.

Para el Admin:

Iniciar sesión: Permite a un administrador iniciar sesión en el sistema.

Agregar Producto: Permite a un administrador agregar nuevos productos al sistema.

Ver Productos: Permite a un administrador visualizar la lista de productos disponibles.

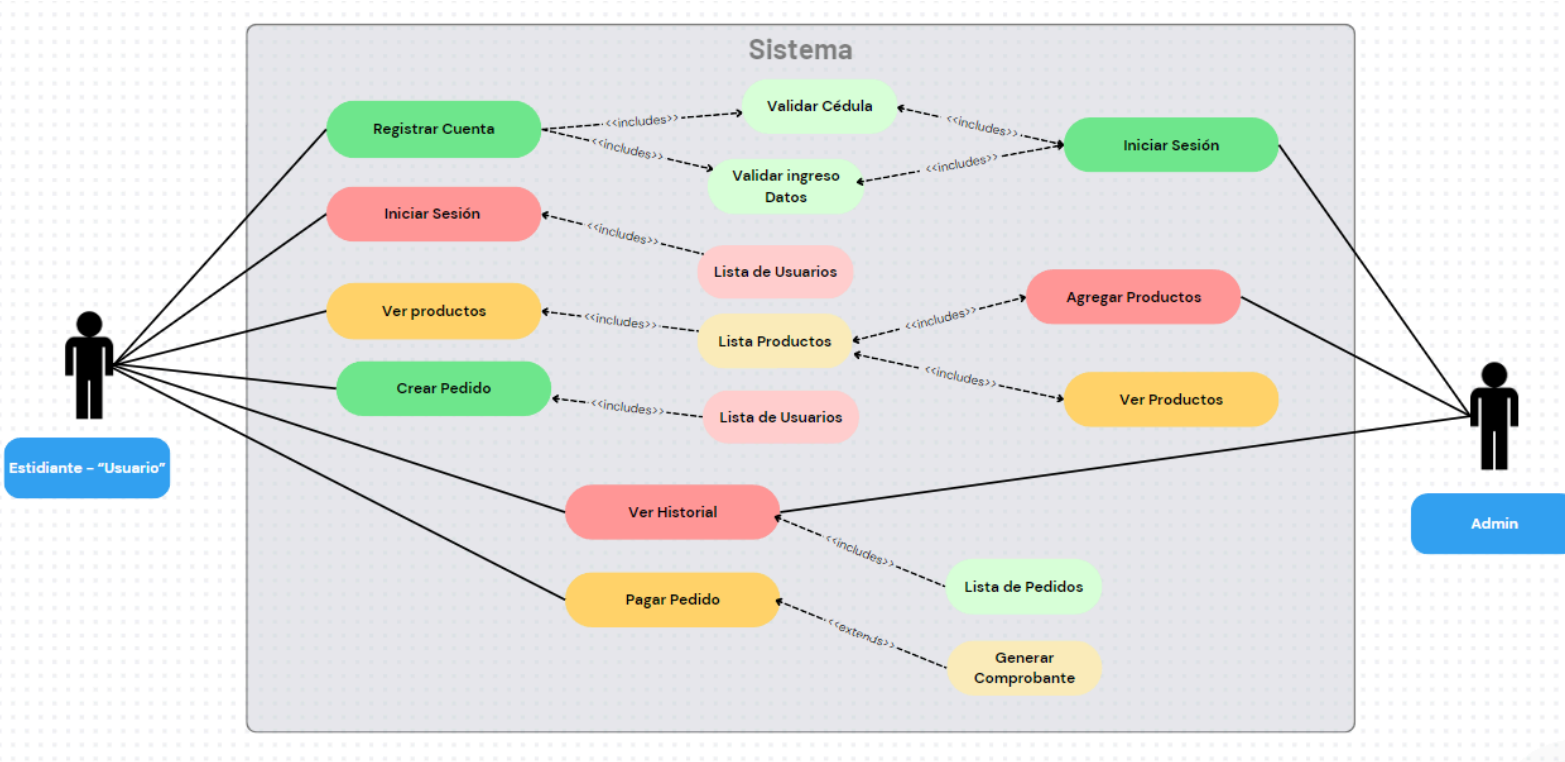
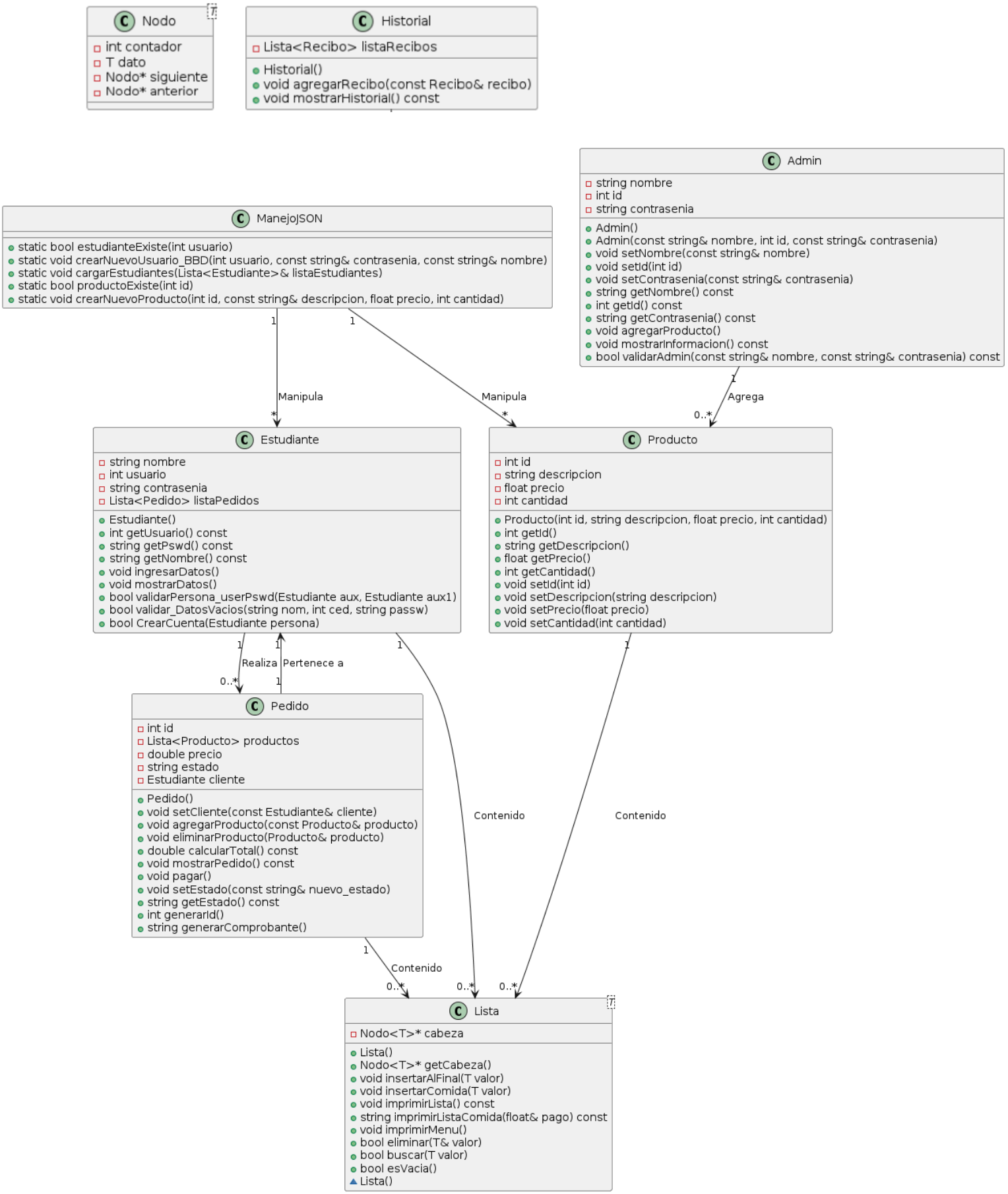


DIAGRAMA DE CLASES


Nodo
int contador
T dato
Nodo* siguiente
Nodo* anterior
Historial
Lista<Recibo> listaRecibos
Historial()
void agregarRecibo(const Recibo& recibo)
void mostrarHistorial() const
Admin
string nombre
int id
string contrasenia
Admin()
Admin(const string& nombre, int id, const string& contrasenia)
void setNombre(const string& nombre)
void setId(int id)
void setContrasenia(const string& contrasenia)
string getNombre() const
int getId() const
string getContrasenia() const
void agregarProducto()
void mostrarInformacion() const
bool validarAdmin(const string& nombre, const string& contrasenia) const
ManejoJSON
static bool estudianteExiste(int usuario)
static void crearNuevoUsuario_BBD(int usuario, const string& contrasenia, const string& nombre)
static void cargarEstudiantes(Lista<Estudiante>& listaEstudiantes)
static bool productoExiste(int id)
static void crearNuevoProducto(int id, const string& descripcion, float precio, int cantidad)
1
Manipula
*
1
Manipula
*
1
Agrega
0..*
Estudiante
string nombre
int usuario
string contrasenia
Lista<Pedido> listaPedidos
Estudiante()
int getUsuario() const
string getPswd() const
string getNombre() const
void ingresarDatos()
void mostrarDatos()
bool validarPersona_userPswd(Estudiante aux, Estudiante aux1)
bool validar_DatosVacios(string nom, int ced, string passw)
bool CrearCuenta(Estudiante persona)
Producto
int id
string descripcion
float precio
int cantidad
Producto(int id, string descripcion, float precio, int cantidad)
int getId()
string getDescripcion()
float getPrecio()
int getCantidad()
void setId(int id)
void setDescripcion(string descripcion)
void setPrecio(float precio)
void setCantidad(int cantidad)
1
Realiza
0..*
1
Pertenece a
1
1
Contenido
0..*
1
Contenido
0..*
Pedido
int id
Lista<Producto> productos
double precio
string estado
Estudiante cliente
Pedido()
void setCliente(const Estudiante& cliente)
void agregarProducto(const Producto& producto)
void eliminarProducto(Producto& producto)
double calcularTotal() const
void mostrarPedido() const
void pagar()
void setEstado(const string& nuevo_estado)
string getEstado() const
int generarId()
string generarComprobante()
1
Contenido
0..*
0..*
0..*
Lista
Nodo<T>* cabeza
Lista()
Nodo<T>* getCabeza()
void insertarAlFinal(T valor)
void insertarComida(T valor)
void imprimirLista() const
string imprimirListaComida(float& pago) const
void imprimirMenu()
bool eliminar(T& valor)
bool buscar(T valor)
bool esVacia()
Lista()

MANEJO DE DATOS

En este sistema manejamos distintas bases de datos que por el momento son bases de dato de tipo texto en formato .Json, para lo que usamos una libreria llamada ***"nlohmann/json.hpp"***

Nlohmann proporciona un conjunto de funcionalidad para trabajar con JSON en C++. Permite el *parsing* (1) y la *serialización* (2) fluida entre objetos C++ y JSON. Es compatible con una variedad de tipos de datos, como cadenas, números y estructuras complejas como arreglos y objetos. Su API intuitiva facilita el acceso y la manipulación de datos JSON de manera eficiente. Además, ofrece una extensa documentación con ejemplos prácticos.
Es compatible con múltiples plataformas, haciéndolo portable para cualquier sistema operativo.

VOCABULARIO

JSON: Es el acrónimo de "JavaScript Object Notation" (Notación de Objetos JavaScript), según *Droettboom. (2015),* Json es un formato simple de intercambio de datos. Surgió como una notación para la World Wide Web y dado que JavaScript está presente en la mayoría de los navegadores web, JSON está basado en JavaScript

Serialización: Es el proceso de convertir datos estructurados o objetos en un formato que pueda ser almacenado o transmitido, y posteriormente reconstruido en su forma original, es decir, implica la transformación de datos complejos en forma lineal o binaria que puede ser fácilmente guardada en archivos o enviada a través de redes de comunicación.

En nuestro contexto la serialización es fundamental para guardar el estado de objetos o estructuras de datos en memoria, en un formato que permita su posterior recuperación y uso.

Cuando trabajamos con JSON, la serialización implica convertir objetos en una cadena de texto JSON que puede ser almacenada en archivos o enviada a través de la red, y luego "*deserializada"* para recuperar el objeto original.

Parsing: Proceso de analizar y descomponer una secuencia de datos o un texto en componentes estructurados que una computadora pueda entender y manipular, es decir, lenguaje máquina.

REFERENCIAS

Borbor Villón, M. C. (2014). Implementación de una aplicación móvil para pedidos de comidas rápidas a domicilio en Italian Gourmet (Bachelor's thesis, La Libertad: Universidad Estatal Península de Santa Elena, 2014o.).

Droettboom, M. (2015). Understanding JSON Schema. Available on: http://spacetelescope. github. io/understanding-jsonschema/UnderstandingJSONSchema. pdf (accessed on 14 April 201)

ANEXOS

En este apartado recopilamos las clases que contienen funcionalidad importante para el manejo de datos, la validación de los mismos y la lógica para su manipulación.

Clase *"ManejoJson.h"*

```
#ifndef MANEJO JSON H
#define MANEJO JSON H
#include <iostream>
#include <fstream>
#include <string>
#include <dirent.h>
#include "nlohmann/json.hpp"
#include "Lista.h"
#include "Estudiante.h"
#include "Producto.h"
using namespace std;
using json = nlohmann::json;
const string CARPETA_USUARIOS = "Usuarios";
const string CARPETA_PRODUCTOS = "Productos";
bool estudianteExiste(int usuario) {
    string nombreArchivo = CARPETA_USUARIOS + "/usuario_" +
to_string(usuario) + ".json";
    ifstream archivo(nombreArchivo);
    return archivo.good();
```

```
}
void crearNuevoUsuario_BBD(int usuario, const string& contrasenia, const
string& nombre) {
    if (estudianteExiste(usuario)) {
        cout << "Error: El estudiante con usuario '" << usuario << "' ya existe."
<< endl;
        return;
    }
    string nombreArchivo = CARPETA_USUARIOS + "/usuario_" +
to_string(usuario) + ".json";

    json j;
    j["ci"] = usuario;
    j["contrasena"] = contrasenia;
    j["nombres"] = nombre;

    ofstream archivo(nombreArchivo);
    if (archivo.is_open()) {
        archivo << j.dump(4);
        archivo.close();
        cout << "Estudiante creado exitosamente: " << nombre << endl;
    } else {
        cerr << "No se pudo abrir el archivo para escribir: " << nombreArchivo
<< endl;
    }
```

```
}
void cargarEstudiantes(Lista<Estudiante>& listaEstudiantes) {
    DIR* dir;
    struct dirent* ent;

    if ((dir = opendir(CARPETA_USUARIOS.c_str())) != NULL) {
        while ((ent = readdir(dir)) != NULL) {
            string nombreArchivo = ent->d_name;
            if (nombreArchivo.find(".json") != string::npos) {
                string rutaArchivo = CARPETA_USUARIOS + "/" + nombreArchivo;
                ifstream archivo(rutaArchivo);
                if (archivo.is_open()) {
                    json j;
                    try {
                        archivo >> j;
                        archivo.close();
                        if (j.contains("ci") && j["ci"].is_number_integer() &&
                            j.contains("contrasena") && j["contrasena"].is_string() &&
                            j.contains("nombres") && j["nombres"].is_string()) {
                            Estudiante estudiante(
                                j["nombres"].get<string>(),
                                j["ci"].get<int>(),
                                j["contrasena"].get<string>()
                            );
                            listaEstudiantes.insertarAlFinal(estudiante);
```

```
                } else {
                    cerr << "El archivo " << rutaArchivo << " no contiene los
campos necesarios o algunos tienen tipos incorrectos." << endl;
                }
            } catch (const json::parse_error& e) {
                cerr << "Error de parseo en el archivo JSON " << rutaArchivo
<< ": " << e.what() << endl;
            } catch (const json::type_error& e) {
                cerr << "Error de tipo en el archivo JSON " << rutaArchivo <<
": " << e.what() << endl;
            } catch (const std::exception& e) {
                cerr << "Error al procesar el archivo JSON " << rutaArchivo
<< ": " << e.what() << endl;
            }
        } else {
            cerr << "No se pudo abrir el archivo: " << rutaArchivo << endl;
        }
    }
    closedir(dir);
  } else {
    cerr << "No se pudo abrir el directorio: " << CARPETA_USUARIOS <<
endl;
  }
}
```

```
bool productoExiste(int id) {
    string nombreArchivo = CARPETA_PRODUCTOS + "/producto_" +
to_string(id) + ".json";
    ifstream archivo(nombreArchivo);
    return archivo.good();
}

void crearNuevoProducto(int id, const string& descripcion, float precio, int
cantidad) {
    if (productoExiste(id)) {
        cout << "Error: El producto con ID '" << id << "' ya existe." << endl;
        return;
    }
    string nombreArchivo = CARPETA_PRODUCTOS + "/producto_" +
to_string(id) + ".json";
    json j;
    j["id"] = id;
    j["descripcion"] = descripcion;
    j["precio"] = precio;
    j["cantidad"] = cantidad;
    ofstream archivo(nombreArchivo);
    if (archivo.is_open()) {
        archivo << j.dump(4);
        archivo.close();
        cout << "Producto creado exitosamente: " << descripcion << endl;
```

```
    } else {
        cerr << "No se pudo abrir el archivo para escribir: " << nombreArchivo
<< endl;
    }
}
#endif
```

Clase *"Validar Cedula.h"*

Esta clase valida el ingreso de cédulas ecuatorianas válidas:

```
#include <iostream>
#include <string>
#include "IngresoDatos.h"
using namespace std;
long long int validarr(long long int &x) {
    IngresoDatos crearUsuario;
    while (x < 100000000 || x > 3999999999) {
        x=crearUsuario.IngresoEnteros("Error, Ingrese de nuevo la cedula consta
de 10 digitos: ");
    }
    return x;
}
long long int validar(long long int &x) {
    IngresoDatos crearUsuario;
    long long int A[10]={0};
    int i = 9, sumapares = 0, sumaimpares = 0, sumat, res, mul, a;
```

```
long long int coc;
long long int aux = x;
x = validarr(x);
   A[0] = 0;
   do {
      coc = aux / 10;
      res = aux % 10;
      A[i] = res;
      aux = coc;
      i--;
   } while (coc != 0);

   for (int j = 0; j < 10; j += 2) {
      mul = A[j] * 2;
      if (mul > 9)
         mul -= 9;
      sumapares += mul;
   }
   for (int j = 1; j < 10 - 1; j += 2)
      sumaimpares += A[j];
   sumat = sumapares + sumaimpares;
   res = 10 - (sumat % 10);
   if (res == 10)
      res = 0;
   if (res == A[9]) {
```

```
        return x;
    }
    else {
        x=crearUsuario.IngresoEnteros("Cedula no valida, ingrese de nuevo:
");
        return validar(x);
    }
}
```

Clase *"ingreso Datos.h"*

Esta clase valida el ingreso de todos los datos que son responsabilidad del usuario, es decir, campos de texto que no sean incorrectos y que correspondan al tipo de dato declarado en el programa para que no falle.

```
#ifndef INGRESODATOS_H
#define INGRESODATOS_H
#include <iostream>
#include <conio.h>
#include <cctype>
#include <string>

using namespace std;

class IngresoDatos{
        public:
```

```
IngresoDatos(){};


int IngresoEnteros(string mensaje){
char *dato=new char[10],c;
int i=0;
cout<<mensaje;
while((c=_getch())!=13){
        if(c>='0'&& c<='9'){
                printf("%c",c);
                dato[i++]=c;
                        }
        else if(c==8||c==127){
                printf("\b \b");
        dato[i--]=0;
        }
}
dato[i]='\0';
cout<<"\n ";
return atoi(dato);
}
string ingresoContra(short minimo, short maximo) {
const char ENTER_KEY = 13;
const char BACKSPACE_KEY = 8;
const char SPACE_KEY = 32;
string password;
```

```
    char keyPressed;
    while (true) {
        keyPressed = _getch();
        if (keyPressed == ENTER_KEY && password.size() >= minimo) {
            break;
        } else if (keyPressed == SPACE_KEY || isalnum(keyPressed)) {
            if (password.size() < maximo) {
                cout << '*';
                password.push_back(keyPressed);
            }
        } else if (keyPressed == BACKSPACE_KEY && !password.empty()) {
            cout << "\b \b";
            password.pop_back();
        }
    }
    // Limpiando espacios en blanco al final de la cadena
    while (!password.empty() && password.back() == SPACE_KEY) {
        password.pop_back();
    }
    cout << '\n';
    return password;
}
\
};
#endif
```