



Universidad Continental

CICLO: III

ASIGNATURA: ESTRUCTURA DE DATOS

NRC: 59098

DOCENTE:

Yesenia Concha Ramos

TRABAJO:

Planificación y diseño

INTEGRANTES:

- Alvares Rodriguez Wernher
- Lima Tintaya Jeferson Jose
- Leguia Choquemamani Patrick Antonio

CUSCO - PERÚ

2025

Índice

1. Descripción del problema.....	3
2. Requerimientos del sistema.....	4
1. Funcionales.....	4
2. No funcionales.....	4
3. Estructuras de datos propuestas.....	5
4. Justificación de la elección.....	5
1. Descripción de estructuras de datos y operaciones:.....	6
1. Estructura: Cada proceso tiene los siguientes atributos.....	6
2. Operaciones principales:.....	6
2. Algoritmos principales:.....	7
a. Pseudocódigo para agregar proceso.....	7
b. Pseudocódigo para cambiar el estado del proceso.....	7
3. Diagramas de Flujo.....	8
Diagrama de proceso de venta.....	9
Diagrama de proceso de postventa.....	10
Diagrama de proceso de servicio técnico.....	11
4. Justificación del diseño:.....	12
1. Código limpio, bien comentado y estructurado.....	13
2. Capturas de pantalla de las ventanas de ejecución con las diversas pruebas de validación de datos.....	18
3. Manual de usuario.....	19
Menú principal.....	19
3 pasos clave para usar el sistema.....	20
Reglas fáciles para ingresar datos.....	20
Errores comunes y solución.....	20
1. Repositorio con Control de Versiones (Capturas de Pantalla).....	21
1.1. Registro de commits claros y significativos que evidencian aportes individuales (proactividad).....	21
1.2. Historial de ramas y fusiones si es aplicable.....	21
1.3. Evidencia por cada integrante del equipo.....	21
1.4. Enlace a la herramienta colaborativa.....	21
2. Plan de Trabajo y Roles Asignados.....	22
2.1. Documento inicial donde se asignan tareas y responsabilidades.....	22
2.2. Cronograma con fechas límite para cada entrega parcial.....	23
2.3. Registro de reuniones o comunicación del equipo (Actas de reuniones.).....	24

CAPÍTULO 1: Análisis del Problema

1. Descripción del problema

Automotriz Cristo Blanco S.A.C., concesionario autorizado de Derco Center en Cusco, se especializa en la comercialización de vehículos, la venta de repuestos originales y la prestación de servicios de mantenimiento preventivo y correctivo. A pesar de su crecimiento sostenido, impulsado por la creciente demanda de soluciones de movilidad en una región turística y económica como Cusco, la empresa enfrenta limitaciones críticas en la gestión interna de sus procesos, particularmente en las áreas de ventas, postventa y servicio técnico, que amenazan su competitividad y capacidad de escalabilidad.

Actualmente, la mayoría de las operaciones se realizan de forma manual, utilizando herramientas como hojas de cálculo, registros físicos y comunicaciones no integradas (correos electrónicos o llamadas telefónicas). Este enfoque genera múltiples problemas operativos:

1. Demoras Significativas

La elaboración de cotizaciones en ventas puede tomar horas o incluso días debido a la verificación manual de inventarios o condiciones de financiamiento, lo que retrasa la respuesta a los clientes y aumenta el riesgo de perder oportunidades frente a competidores más ágiles. En postventa, la programación de citas y el seguimiento de mantenimientos dependen de agendas físicas, lo que provoca esperas prolongadas para los clientes, con tiempos de respuesta que pueden extenderse hasta 48 horas o más.

2. Errores de Coordinación

La falta de un sistema centralizado genera duplicidad en los registros, como contactar repetidamente a un mismo cliente por parte de distintos vendedores o programar citas técnicas conflictivas. En el área de servicio técnico, los errores en la gestión manual de inventarios de repuestos ocasionan faltantes críticos o acumulación de stock obsoleto, lo que puede incrementar costos operativos en un 15-20% y alargar los tiempos de reparación, afectando la experiencia del cliente.

3. Falta de Control de Información

Sin una base de datos unificada, la empresa no puede rastrear en tiempo real datos clave, como el historial de compras de un cliente, el estado de garantías o el desempeño de las ventas. Esto resulta en pérdida de información valiosa, como olvidar notificar a un cliente sobre un mantenimiento programado, o en errores que afectan el cumplimiento de normativas, como la Ley de Protección de Datos Personales en Perú. Por ejemplo, un cliente podría recibir información inconsistente sobre el estado de su vehículo, lo que genera desconfianza.

Estas ineficiencias impactan directamente la atención al cliente, provocando insatisfacción reflejada en quejas frecuentes, tiempos de espera prolongados y una menor tasa de retención, estimada en una posible disminución del 10-15% en clientes recurrentes. Además, la incapacidad de generar reportes en tiempo real limita la toma de decisiones estratégicas, como identificar tendencias de ventas o gestionar eficientemente el inventario, lo que podría traducirse en pérdidas financieras significativas, como miles de soles mensuales en oportunidades de venta desaprovechadas. En un mercado automotriz cada vez más competitivo y digitalizado, estas limitaciones posicionan a Automotriz Cristo Blanco S.A.C. en desventaja frente a competidores que ya implementan soluciones tecnológicas integradas.

2. Requerimientos del sistema

1. Funcionales

- Registrar clientes, vehículos, repuestos y servicios.
- Generar cotizaciones, boletas y facturas.
- Controlar inventarios y mantenimientos.
- Programar citas y asignar técnicos.
- Emitir reportes automáticos de ventas y servicios.
- Notificar al cliente sobre entregas o garantías.

2. No funcionales

- Interfaz sencilla y accesible para todo el personal.
- Base de datos segura y con respaldo.
- Acceso simultáneo desde distintas áreas.
- Protección de datos personales.
- Sistema modular, rápido y escalable.

3. Estructuras de datos propuestas

Módulo	Estructura de datos	Descripción
Registro de Clientes y Vehículos	Lista enlazada	Permite almacenar y actualizar la información de clientes, vehículos y repuestos de forma dinámica. Facilita la inserción, búsqueda y eliminación de registros sin necesidad de un tamaño fijo.
Gestión de Servicios Técnicos	Cola de prioridad	Organiza las solicitudes de servicio según su nivel urgencia (garantías, mantenimientos, reparaciones). Permite una atención ordenada y eficiente de los casos.
Control de Inventario (Repuestos)	Pila	Administra el uso y reposición de repuestos en el taller aplicando el principio LIFO (último en

LIFO: (*Last In, First Out*) significa “Último en entrar, primero en salir”.

Es un método donde el último elemento que se agrega es el primero que se retira.

Se aplica en pilas de datos o en inventarios, como en un taller automotriz donde los repuestos más recientes se usan primero.

4. Justificación de la elección

La decisión de tratar las limitaciones en la gestión de Automotriz Cristo Blanco S. A. C. a través de un Sistema Integral de Gestión Automotriz (SGAI) se fundamenta en su efectividad para abordar fallas significativas en ventas, servicio postventa y atención técnica, mejorando la eficiencia operativa y elevando el servicio al cliente en el competitivo mercado de Cusco. La elección de las estructuras de datos sugeridas, representadas mediante diagramas de flujo, se asocia de manera óptima con los procesos específicos identificados.

- Para el **Registro de Clientes y Vehículos**, la lista enlazada permite una gestión dinámica de la información, facilitando la inserción, búsqueda y eliminación sin requerir un tamaño predeterminado, lo que agiliza la administración de cotizaciones e históricas en flujos secuenciales.
- En la **Gestión de Servicios Técnicos**, la cola de prioridad clasifica las solicitudes (garantías, mantenimientos, reparaciones) según su urgencia, lo que asegura una atención organizada y efectiva, reduciendo los tiempos de espera y mejorando la satisfacción de los clientes.
- Para el **Control de Inventario (Repuestos)**, la pila sigue un enfoque LIFO (Último en entrar, primero en salir), optimizando la administración de repuestos al sacar el más reciente primero, lo que reduce desperdicios y acelera la rotación de inventario en diagramas de flujo claramente definidos.

Estas estructuras, creadas para diagramas de flujo, aseguran simplicidad, rapidez y capacidad de adaptación, permitiendo que el SGAI se ajuste al crecimiento de la empresa sin sacrificar la calidad del servicio. Con su implementación, se eliminan ineficiencias operativas, se respetan normativas como la Ley de Protección de Datos Personales y se establece a Automotriz Cristo Blanco S. A. C. como un líder innovador en un mercado en expansión.

CAPÍTULO 2: Diseño de la Solución

1. Descripción de estructuras de datos y operaciones:

1. Estructura: Cada proceso tiene los siguientes atributos

El sistema propuesto para Automotriz Cristo Blanco S.A.C. Se basa en tres módulos principales: ventas, postventa y servicio técnico. Cada uno utiliza estructuras dinámicas que permiten manejar la información de manera flexible y eficiente.

- En el módulo de ventas, se emplea una lista enlazada para registrar clientes, vehículos y transacciones.
- En postventa, se utiliza una cola de prioridad para gestionar reclamos y servicios según su urgencia.
- En servicio técnico, se implementa una pila para administrar el uso y reposición de repuestos aplicando el principio LIFO.

Estas estructuras permiten que el sistema funcione de forma dinámica, optimizando tiempos de búsqueda, registro y control de recursos.

2. Operaciones principales:

El Sistema de Gestión Automotriz Integral (SGAI) para Automotriz Cristo Blanco S.A.C. incluye un conjunto de operaciones principales diseñadas para optimizar los procesos en los módulos de ventas, postventa y servicio técnico. A continuación, se detallan las operaciones y su descripción específica por módulo, ampliando la funcionalidad para cubrir necesidades operativas adicionales:

- **Módulo de Ventas:**

- **RegistrarCliente():** Registra un nuevo cliente en la lista enlazada, capturando datos esenciales como nombre, contacto, historial de compras y preferencias. Esta operación inicializa un nodo único en la estructura, facilitando su uso en futuras transacciones y consultas.
- **RegistrarVehículo():** Ingresa detalles de un vehículo (modelo, año, precio, estado) en la lista enlazada, actualizando el inventario disponible y permitiendo su asociación con clientes potenciales o ventas en curso.
- **GenerarVenta():** Finaliza una transacción de venta, vinculando un cliente y un vehículo de la lista enlazada, actualizando el stock, generando un comprobante digital y registrando el historial de la venta para análisis posteriores.
- **ConsultarHistorialCliente():** Permite revisar el historial de compras y servicios de un cliente específico, extrayendo datos de la lista enlazada para ofrecer recomendaciones personalizadas o verificar garantías.

- **Módulo de Postventa:**

- **EncolarServicio():** Agrega un servicio (reclamo, garantía, mantenimiento) a la cola de prioridad, asignándole un nivel de urgencia y una fecha de registro para su procesamiento ordenado según prioridad.
- **EjecutarServicio():** Inicia la atención de un servicio extraído de la cola de prioridad, actualizando su estado y asignando recursos (técnicos, repuestos) para su resolución.
- **CambiarEstadoServicio():** Modifica el estado de un servicio (pendiente, en proceso, finalizado) en la cola, notificando al cliente y actualizando los registros para un seguimiento transparente.
- **NotificarCliente():** Envía una alerta automática (vía correo o SMS) al cliente sobre el estado de su servicio o vencimiento de garantías, mejorando la comunicación postventa.
- **Módulo de Servicio Técnico:**
 - **AsignarRepuesto():** Extrae el repuesto más reciente de la pila (LIFO) para su uso en una reparación, registrando su salida del inventario y actualizando el stock disponible.
 - **LiberarRepuesto():** Reingresa un repuesto a la pila, ya sea por devolución o reposición, actualizando el inventario y asegurando la continuidad del suministro.
 - **EjecutarServicio():** Procesa un servicio técnico asignado, utilizando repuestos de la pila y actualizando el estado del servicio en la cola de prioridad para reflejar su progreso.
 - **VerificarInventario():** Revisa el estado del inventario de repuestos en la pila, identificando niveles críticos o vencimientos para planificar reposiciones oportunas.

2. Algoritmos principales:

a. Pseudocódigo para agregar proceso

```

PROCEDIMIENTO AgregarProceso(id_cliente, tipo_proceso, prioridad)
    NUEVO ← CREAR_NODO()
    NUEVO.id_proceso ← generarID()
    NUEVO.id_cliente ← id_cliente
    NUEVO.tipo_proceso ← tipo_proceso
    NUEVO.prioridad ← prioridad
    NUEVO.estado ← "Pendiente"
    NUEVO.fecha ← fecha_actual()
    ENCOLAR(cola_procesos, NUEVO)
    MOSTRAR "Proceso registrado correctamente para el cliente: " +
        id_cliente
FIN PROCEDIMIENTO

```

b. Pseudocódigo para cambiar el estado del proceso

```

PROCEDIMIENTO CambiarEstadoProceso(id_proceso, nuevo_estado)
    TEMP ← BUSCAR(cola_procesos, id_proceso)
    SI TEMP ≠ NULL ENTONCES

```

```

TEMP.estado ← nuevo_estado
MOSTRAR "El estado del proceso ha sido actualizado a: " +
nuevo_estado
SINO
    MOSTRAR "Proceso no encontrado en el sistema."
FIN SI
FIN PROCEDIMIENTO

```

3. Diagramas de Flujo

Diagrama de proceso de venta

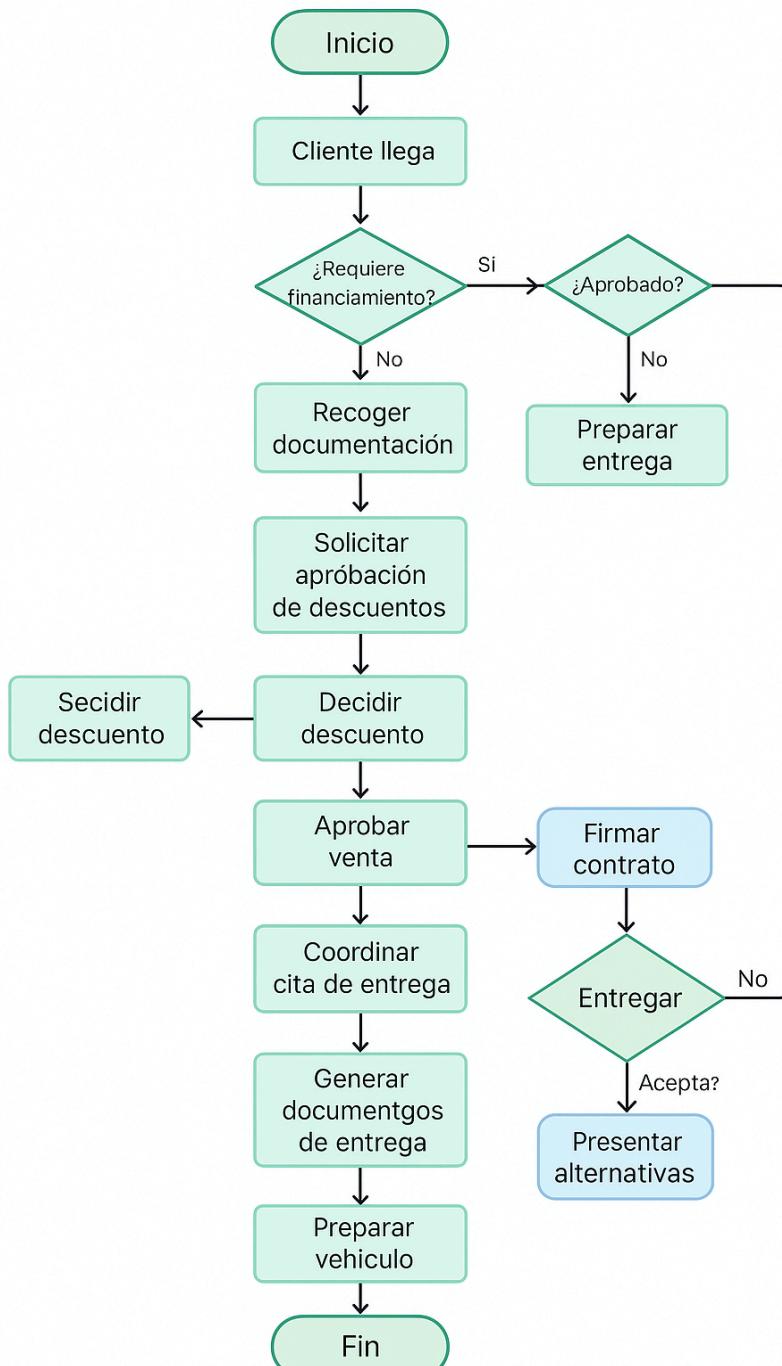


Diagrama de proceso de postventa.

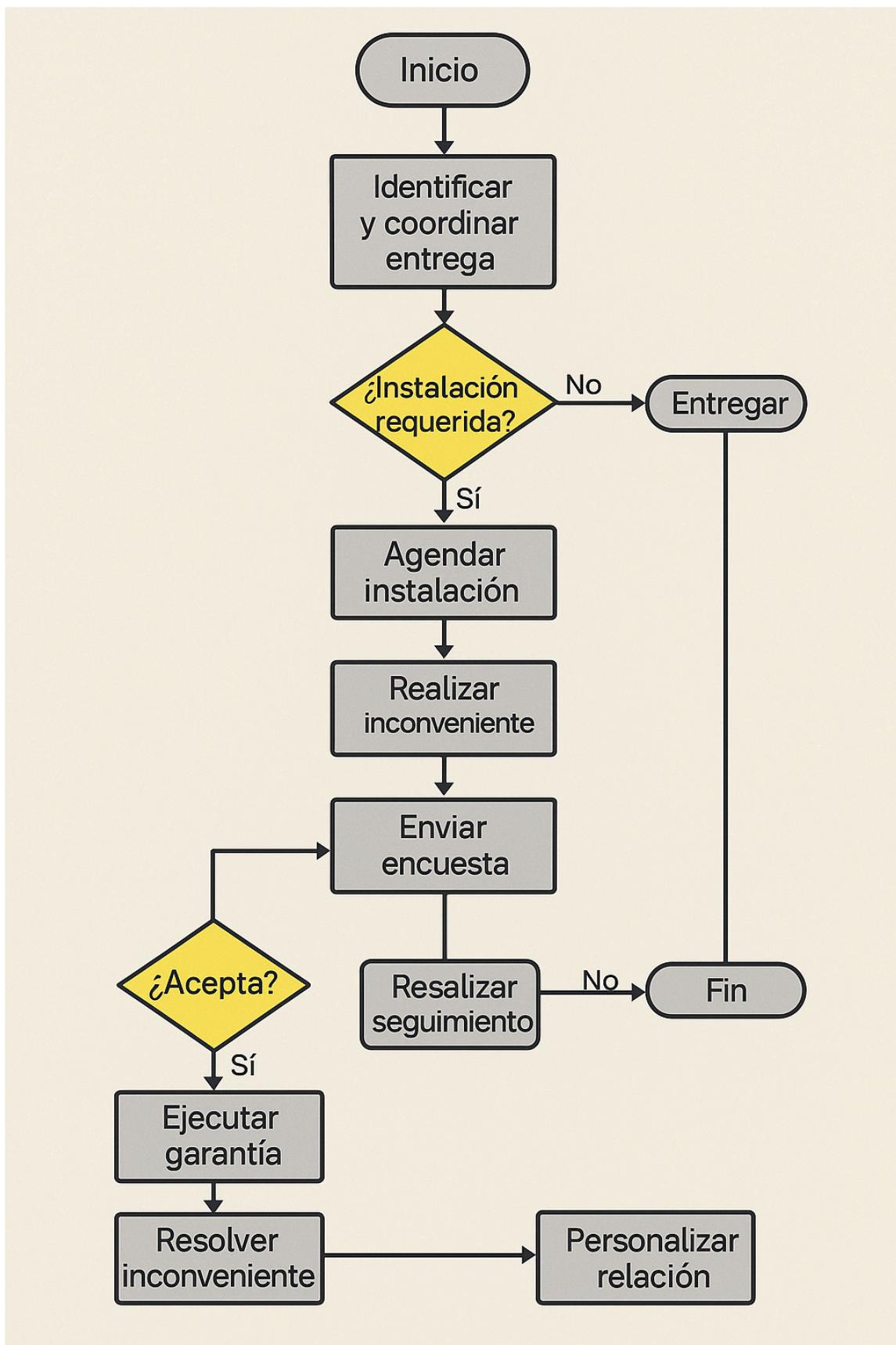
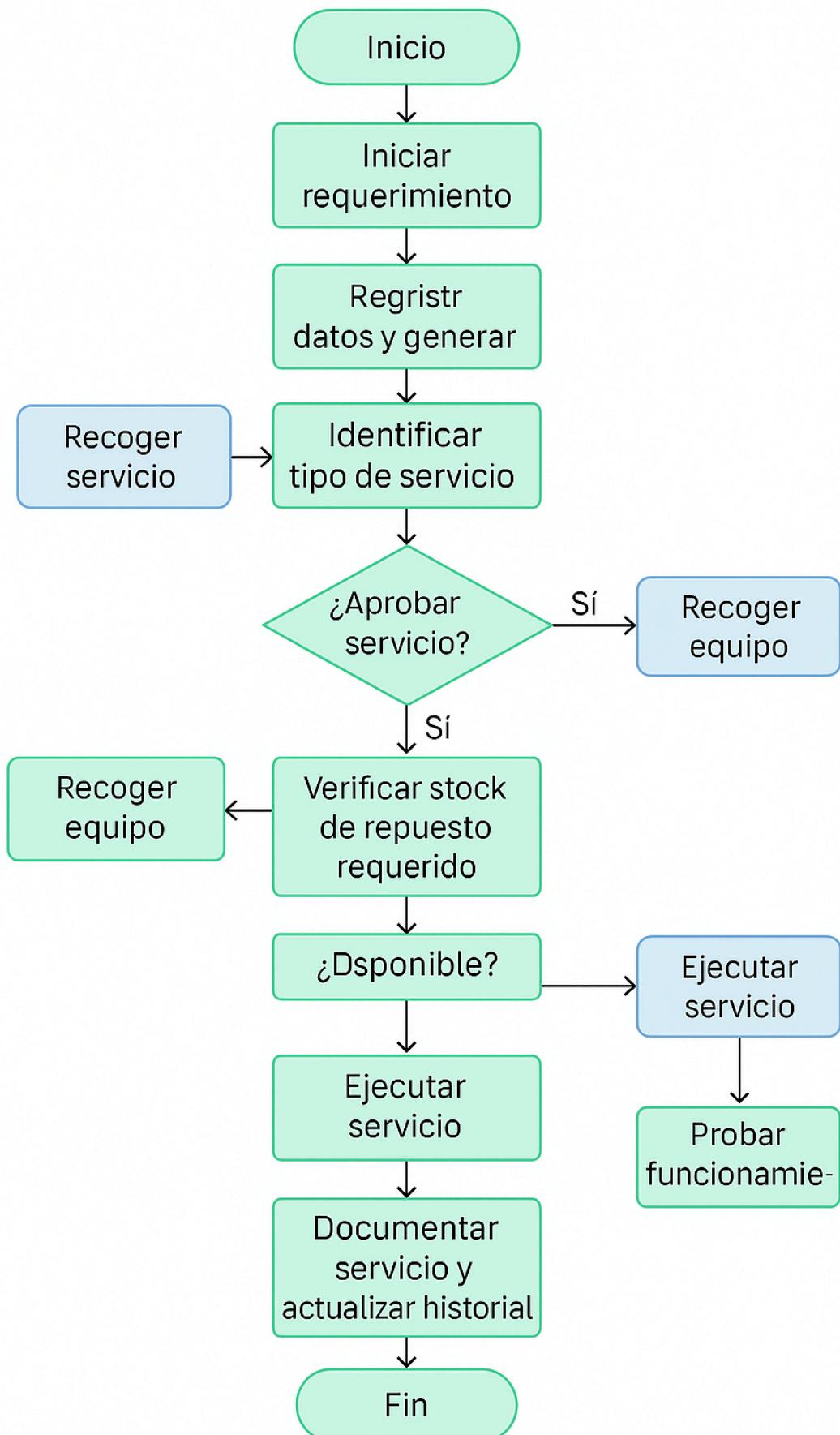


Diagrama de proceso de servicio técnico



4. Justificación del diseño:

El diseño del Sistema de Gestión Automotriz Integral (SGAI) para Automotriz Cristo Blanco S.A.C. se fundamenta en una arquitectura modular y escalable que integra las áreas críticas de ventas, postventa y servicio técnico mediante una base de datos relacional centralizada. Esta arquitectura emplea un esquema normalizado con claves primarias y foráneas para garantizar la integridad referencial entre entidades (clientes, vehículos, transacciones, repuestos), eliminando redundancias y asegurando consistencia a través de transacciones controladas con aislamiento y durabilidad. La implementación utiliza técnicas de persistencia con journaling para la recuperación ante fallos, un requisito esencial en un entorno de alta transaccionalidad como el de un concesionario.

El diseño incorpora estructuras dinámicas optimizadas para cada módulo: en el módulo de ventas, listas enlazadas permiten la gestión dinámica de registros mediante nodos encadenados, soportando inserciones y eliminaciones con acceso directo a través de punteros, lo que facilita la escalabilidad sin reestructuración previa. En postventa, una cola de prioridad implementada con un heap jerárquico organiza solicitudes (reclamos, garantías) según niveles de urgencia, empleando un algoritmo de reordenamiento basado en comparaciones para priorizar en tiempo real. En servicio técnico, una pila adopta el principio LIFO (Last In, First Out) con un enfoque de última entrada, primera salida, utilizando un mecanismo de puntero de cima para gestionar el inventario de repuestos, optimizando la rotación mediante un protocolo de entrada/salida secuencial.

Estas estructuras se apoyan en técnicas avanzadas de manipulación de datos: las listas enlazadas emplean un sistema de navegación por nodos para búsquedas y actualizaciones eficientes; las colas de prioridad integran un mecanismo de encolamiento/descolamiento basado en prioridad dinámica; y las pilas utilizan un protocolo de apilamiento/desapilamiento con control de puntero para un acceso inmediato al elemento superior. Estas soluciones aseguran flexibilidad operativa, escalabilidad vertical y horizontal, y trazabilidad mediante logs transaccionales que registran cada operación con sellos temporales, cumpliendo con estándares de auditoría y normativas como la Ley de Protección de Datos Personales en Perú. La seguridad se refuerza con autenticación multifactor y cifrado AES-256 en la base de datos, protegiendo datos sensibles como información de clientes y transacciones.

Capítulo 3: Solución Final

1. Código limpio, bien comentado y estructurado.

```
// === ESTRUCTURA CLIENTE ===
struct Cliente {
    int id;           // ID numérico único (DNI)
    string nombre;    // Nombre del cliente
    string contacto; // Teléfono
    Cliente* siguiente; // Puntero al siguiente nodo en la lista enlazada
};
Cliente* cabezaClientes = NULL; // Apunta al primer cliente (lista vacía al inicio)

// === REGISTRAR CLIENTE ===
// Agrega un nuevo cliente al final de la lista enlazada
void registrarCliente(int id, string nombre, string contacto) {
    Cliente* nuevo = new Cliente; // Crea un nuevo nodo en memoria dinámica
    nuevo->id = id;
    nuevo->nombre = nombre;
    nuevo->contacto = contacto;
    nuevo->siguiente = NULL;

    if (cabezaClientes == NULL) {
        cabezaClientes = nuevo; // Si la lista está vacía, el nuevo es el primero
    } else {
        Cliente* temp = cabezaClientes;
        while (temp->siguiente != NULL) {
            temp = temp->siguiente; // Recorre hasta el último nodo
        }
        temp->siguiente = nuevo; // Enlaza el nuevo nodo al final
    }
    cout << "Cliente registrado: " << nombre << endl;
}
```

```
// === MOSTRAR CLIENTES ===
// Recorre e imprime todos los clientes registrados
void mostrarClientes() {
    Cliente* temp = cabezaClientes;
    if (temp == NULL) {
        cout << "No hay clientes registrados.\n";
        return;
    }
    cout << "\n--- Lista de Clientes ---\n";
    while (temp != NULL) {
        cout << "ID: " << temp->id
            << " | Nombre: " << temp->nombre
            << " | Contacto: " << temp->contacto << endl;
        temp = temp->siguiente;
    }
}
```

```

// === ESTRUCTURA SERVICIO ===
// Cola con prioridad: 1 = alta, 2 = media, 3 = baja
struct Servicio {
    int id;
    string tipo;
    int prioridad; // Menor número = mayor prioridad
    Servicio* siguiente;
};
Servicio* frenteServicios = NULL; // Frente de la cola (primer servicio)

// === ENCOLAR SERVICIO ===
// Inserta un servicio ordenado por prioridad (menor número primero)
void encolarServicio(int id, string tipo, int prioridad) {
    Servicio* nuevo = new Servicio{id, tipo, prioridad, NULL};

    if (frenteServicios == NULL || prioridad < frenteServicios->prioridad) {
        nuevo->siguiente = frenteServicios;
        frenteServicios = nuevo; // Inserta al inicio si tiene mayor prioridad
    } else {
        Servicio* temp = frenteServicios;
        while (temp->siguiente != NULL && temp->siguiente->prioridad <= prioridad) {
            temp = temp->siguiente;
        }
        nuevo->siguiente = temp->siguiente;
        temp->siguiente = nuevo;
    }
    cout << "Servicio agregado: " << tipo << " (Prioridad " << prioridad << ")\n";
}

```

```

// === EJECUTAR SERVICIO ===
// Elimina y ejecuta el servicio de mayor prioridad (frente de la cola)
void ejecutarServicio() {
    if (frenteServicios == NULL) {
        cout << "No hay servicios pendientes.\n";
        return;
    }
    Servicio* temp = frenteServicios;
    frenteServicios = frenteServicios->siguiente;
    cout << "Ejecutando servicio ID " << temp->id << ": " << temp->tipo << endl;
    delete temp; // Libera memoria
}

```

```

// === MOSTRAR SERVICIOS ===
// Muestra todos los servicios en orden de prioridad
void mostrarServicios() {
    if (frenteServicios == NULL) {
        cout << "No hay servicios en cola.\n";
        return;
    }
    Servicio* temp = frenteServicios;
    cout << "\n--- Cola de Servicios (Postventa) ---\n";
    while (temp != NULL) {
        cout << "ID: " << temp->id
            << " | Tipo: " << temp->tipo
            << " | Prioridad: " << temp->prioridad << endl;
        temp = temp->siguiente;
    }
}

```

```

// === ESTRUCTURA REPUESTO ===
// Pila (LIFO): último en entrar, primero en salir
struct Repuesto {
    int id;
    string nombre;
    Repuesto* abajo; // Apunta al repuesto anterior (debajo en la pila)
};
Repuesto* cimaRepuestos = NULL; // Cima de la pila

// === AGREGAR REPUESTO ===
// Empuja un nuevo repuesto a la cima de la pila
void agregarRepuesto(int id, string nombre) {
    Repuesto* nuevo = new Repuesto{id, nombre, cimaRepuestos};
    cimaRepuestos = nuevo;
    cout << "Repuesto agregado: " << nombre << endl;
}

// === USAR REPUESTO ===
// Saca y usa el repuesto de la cima
void usarRepuesto() {
    if (cimaRepuestos == NULL) {
        cout << "No hay repuestos disponibles.\n";
        return;
    }
    Repuesto* temp = cimaRepuestos;
    cimaRepuestos = cimaRepuestos->abajo;
    cout << "Usando repuesto: " << temp->nombre << endl;
    delete temp;
}

// === MOSTRAR REPUESTOS ===
// Muestra la pila desde la cima hacia abajo
void mostrarRepuestos() {
    if (cimaRepuestos == NULL) {
        cout << "Inventario vacío.\n";
        return;
    }
    Repuesto* temp = cimaRepuestos;
    cout << "\n--- Pila de Repuestos ---\n";
    while (temp != NULL) {
        cout << "ID: " << temp->id << " | Nombre: " << temp->nombre << endl;
        temp = temp->abajo;
    }
}

// Devuelve true si el ID existe, false si no
bool existeCliente(int idBuscado) {
    Cliente* temp = cabezaClientes;
    while (temp != NULL) {
        if (temp->id == idBuscado) {
            return true;
        }
        temp = temp->siguiente;
    }
    return false;
}

// Devuelve el nombre si existe, "" si no
string obtenerNombreCliente(int idBuscado) {
    Cliente* temp = cabezaClientes;
    while (temp != NULL) {
        if (temp->id == idBuscado) {
            return temp->nombre;
        }
        temp = temp->siguiente;
    }
    return "";
}

```

```

// === REGISTRAR CLIENTE CON VALIDACIÓN ===
case 1: {
    int id;
    string nombre, contacto;

    // ID Cliente: (DNI)
    cout << "ID Cliente (ingrese manualmente): ";
    while (true) {
        if (cin >> id && id > 0) {
            // Verificar que el ID no esté ya registrado
            if (!existeCliente(id)) {
                cin.ignore(10000, '\n');
                break;
            } else {
                cout << "Error: El ID " << id << " ya está registrado. Ingrese otro: ";
            }
        } else {
            cout << "Error:Ingrese solo numeros mayores a 0: ";
            cin.clear();
            cin.ignore(10000, '\n');
        }
    }

    // NOMBRE: solo letras y espacios
    cout << "Nombre (solo letras): ";
    while (true) {
        getline(cin, nombre);
        if (nombre.empty()) {
            cout << "Error: No puede estar vacío: ";
            continue;
        }
        bool ok = true;
        for (char c : nombre) {
            if (!(c >= 'a' && c <= 'z') ||
                (c >= 'A' && c <= 'Z') ||
                c == ' ') {
                ok = false;
                break;
            }
        }
        if (ok) break;
        cout << "Error: Solo letras y espacios: ";
    }
}

// CONTACTO: solo numeros
cout << "Contacto (solo numeros): ";
while (true) {
    getline(cin, contacto);
    if (contacto.empty()) {
        cout << "Error: No puede estar vacío: ";
        continue;
    }
    bool ok = true;
    for (char c : contacto) {
        if (c < '0' || c > '9') {
            ok = false;
            break;
        }
    }
    if (ok) break;
    cout << "Error: Solo numeros permitidos: ";
}

registrarCliente(id, nombre, contacto);
break;
}

```

```

case 3: {
    // Verificar si hay clientes registrados
    if (cabezaClientes == NULL) {
        cout << "No hay clientes registrados. Registre al menos un cliente primero.\n";
        break;
    }

    int idCliente, prioridad;
    string tipo;

    // ID Cliente: debe existir
    cout << "ID Cliente (registrado): ";
    while (true) {
        if (cin >> idCliente && idCliente > 0) {
            if (existeCliente(idCliente)) {
                cin.ignore(10000, '\n');
                break;
            } else {
                cout << "Error: No existe cliente con ID " << idCliente << ". Intente otro: ";
            }
        } else {
            cout << "Error: Ingrese solo numeros mayores a 0: ";
            cin.clear();
            cin.ignore(10000, '\n');
        }
    }

    // ID Servicio: automático
    int idServicio = contadorServicios++;
    cout << "ID Servicio (autogenerado): " << idServicio << endl;

    // Tipo de servicio
    cout << "Tipo de servicio: ";
    getline(cin, tipo);
    if (tipo.empty()) {
        tipo = "Sin descripcion";
    }

    // Prioridad: 1, 2 o 3
    cout << "Prioridad (1=Alta, 2=Media, 3=Baja): ";
    while (!(cin >> prioridad) || prioridad < 1 || prioridad > 3) {
        cout << "Error: Prioridad debe ser 1, 2 o 3: ";
        cin.clear();
        cin.ignore(10000, '\n');
    }
    cin.ignore(); // limpia el salto de linea después de prioridad

    string nombreCliente = obtenerNombreCliente(idCliente);
    cout << "Servicio para: " << nombreCliente << endl;
    encolarServicio(idServicio, tipo + " (Cliente ID: " + to_string(idCliente) + ")", prioridad);
    break;
}

```

```

case 4: mostrarServicios(); break;
case 5: ejecutarServicio(); break;
case 6: {
    // Verificar si hay clientes registrados
    if (cabezaClientes == NULL) {
        cout << "No hay clientes registrados. Registre al menos un cliente primero.\n";
        break;
    }

    int idCliente;
    string nombre;

    // ID Cliente: debe existir
    cout << "ID Cliente (registrado): ";
    while (true) {
        if (cin >> idCliente && idCliente > 0) {
            if (existeCliente(idCliente)) {
                cin.ignore(10000, '\n');
                break;
            } else {
                cout << "Error: No existe cliente con ID " << idCliente << ". Intente otro: ";
            }
        } else {
            cout << "Error: Ingrese solo numeros mayores a 0: ";
            cin.clear();
            cin.ignore(10000, '\n');
        }
    }

    // ID Repuesto: automático
    int idRepuesto = contadorRepuestos++;
    cout << "ID Repuesto (autogenerado): " << idRepuesto << endl;

    // Nombre del repuesto
    cout << "Nombre del repuesto: ";
    getline(cin, nombre);
    if (nombre.empty()) {
        nombre = "Sin nombre";
    }

    // Mostrar información ANTES de agregar
    string nombreCliente = obtenerNombreCliente(idCliente);
    cout << "Repuesto para: " << nombreCliente << endl;

    // Agregar al sistema
    agregarRepuesto(idRepuesto, nombre + " (Cliente ID: " + to_string(idCliente) + ")");
    break;
}
case 7: mostrarRepuestos(); break;
case 8: usarRepuesto(); break;
case 9: cout << "Saliendo..." << endl; break;
default: cout << "Opcion no valida. Intente nuevamente.\n"; break;
}

```

2. Capturas de pantalla de las ventanas de ejecución con las diversas pruebas de validación de datos

Seleccione una opcion: 1

ID Cliente (ingrese manualmente): 76532123

Nombre (solo letras): Juan Perez

Contacto (solo numeros): 982374232

Cliente registrado: Juan Perez

Seleccione una opcion: 2

--- Lista de Clientes ---

ID: 76532123 | Nombre: Juan Perez | Contacto: 982374232

```
Seleccione una opcion: 3

ID Cliente (registrado): 76532123
ID Servicio (autogenerado): 1
Tipo de servicio: Mejoramiento
Prioridad (1=Alta, 2=Media, 3=Baja): 1
Servicio para: Juan Perez
Servicio agregado: Mejoramiento (Cliente ID: 76532123) (Prioridad 1)

Seleccione una opcion: 4

--- Cola de Servicios (Postventa) ---
ID: 1 | Tipo: Mejoramiento (Cliente ID: 76532123) | Prioridad: 1

Seleccione una opcion: 5

Ejecutando servicio ID 1: Mejoramiento (Cliente ID: 76532123)

Seleccione una opcion: 6

ID Cliente (registrado): 76532123
ID Repuesto (autogenerado): 1
Nombre del repuesto: Bateria
Repuesto para: Juan Perez
Repuesto agregado: Bateria (Cliente ID: 76532123)

Seleccione una opcion: 7

--- Pila de Repuestos ---
ID: 1 | Nombre: Bateria (Cliente ID: 76532123)

Seleccione una opcion: 8

Usando repuesto: Bateria (Cliente ID: 76532123)

Seleccione una opcion: 9

Saliendo ...
```

3. Manual de usuario

Menú principal

Al iniciar el programa, verás estas 9 opciones exactas para que puedas manejar todo el taller:

- Registrar cliente (Ventas)

- Mostrar clientes
- Agregar servicio (Postventa)
- Mostrar servicios
- Ejecutar servicio
- Agregar repuesto (Servicio Técnico)
- Mostrar repuestos
- Usar repuesto
- Salir

3 pasos clave para usar el sistema

Aquí te explicamos con ejemplos sencillos cómo usar las opciones más importantes:

- **Opción 1: Registrar cliente**

Ejemplo: *ID: 12345678 → Nombre: Juan Pérez → Teléfono: 987654321*

Introduce el ID, nombre y teléfono para guardar un nuevo cliente.

- **Opción 3: Agregar servicio**

Ejemplo: *ID Cliente: 12345678 → Tipo: Cambio de aceite → Prioridad: 1*

Asigna un servicio al cliente con el tipo y la prioridad (1 alta, 2 media, 3 baja).

- **Opción 6: Agregar repuesto**

Ejemplo: *ID Cliente: 12345678 → Repuesto: Llantas*

Añade repuestos necesarios a los clientes para el servicio técnico.

Reglas fáciles para ingresar datos

- **ID cliente:**

Debe ser un número único, no repetirlo para otro cliente.

- **Nombre:**

Solo letras, sin números ni símbolos.

- **Teléfono:**

Solo números, sin letras ni espacios.

- **Prioridad:**

Solo puede ser 1, 2 o 3.

Errores comunes y solución

Error	Solución
No hay clientes registrados	Usa la opción 1 para registrar clientes antes

Cuando completes una acción, **presiona Enter para continuar** y seguir usando el sistema sin problemas.

Capítulo 4: Evidencias de Trabajo en Equipo

1. Repositorio con Control de Versiones (Capturas de Pantalla)

1.1. Registro de commits claros y significativos que evidencian aportes individuales (proactividad).

The screenshot shows a GitHub repository interface. At the top, there's a navigation bar with links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main area is titled "Commits" and shows a dropdown menu for the branch "main". Below this, a search bar has "All users" and "All time" selected. A list of commits is displayed, each with a small profile icon, the author's name (Tomm1), the commit message, and the date it was made (e.g., "2 hours ago"). The commits are:

- Solucion de ID en programa (Tomm1 committed 25 minutes ago)
- error en registro arreglado (Tomm1 committed 2 hours ago)
- Codigo Comentado (Tomm1 committed 2 hours ago)
- Solo numeros en ID y Contacto (Tomm1 committed 2 hours ago)
- Mejora de menu (Tomm1 committed 3 hours ago)
- Menu de opciones (Tomm1 committed 3 hours ago)
- Añadicion de Modulo de Repuestos (Tomm1 committed 3 hours ago)
- Ejecutar y Mostrar Servicios (Tomm1 committed 3 hours ago)
- Encolamiento de Servicios (Tomm1 committed 3 hours ago)
- Metodo MostrarClientes (Tomm1 committed 3 hours ago)
- Registrar Cliente (Tomm1 committed 3 hours ago)

1.2. Historial de ramas y fusiones si es aplicable.

```
Desde https://github.com/Patrick1242/Proyecto_Estructura_de_Datos_UC
 * branch           main      → FETCH_HEAD
Ya está actualizado.
```

```
* main
remotes/origin/main
```

1.3. Evidencia por cada integrante del equipo.

Patrick Leguia (Creacion de el repositorio en Github)

1.4. Enlace a la herramienta colaborativa

https://github.com/Patrick1242/Proyecto_Estructura_de_Datos_UC

2. Plan de Trabajo y Roles Asignados

2.1. Documento inicial donde se asignan tareas y responsabilidades.

Capítulos	Actividades	Responsable	Tareas y Responsabilidades
CAPÍTULO 1: Análisis del Problema	1. Descripción del problema	Jeferson	- Redactar una descripción clara del problema a resolver.- Identificar contexto, usuarios y limitaciones.- Definir objetivos principales del sistema.
	2. Requerimientos del sistema	Patrick	- Detallar requerimientos funcionales y no funcionales .- Verificar coherencia con los objetivos del sistema.- Clasificar requerimientos según prioridad.
	3. Estructuras de datos propuestas	Jeferson	- Proponer estructuras de datos adecuadas al problema.- Describir su función dentro del sistema.- Identificar relaciones entre datos.
	4. Justificación de la elección	Patrick	- Argumentar por qué las estructuras elegidas son las más convenientes.- Considerar eficiencia, facilidad de implementación y mantenimiento.
CAPÍTULO 2: Diseño de la Solución	1. Descripción de estructuras de datos y operaciones	Jeferson	- Describir en detalle las estructuras de datos implementadas.- Explicar operaciones básicas (agregar, eliminar, buscar, actualizar).

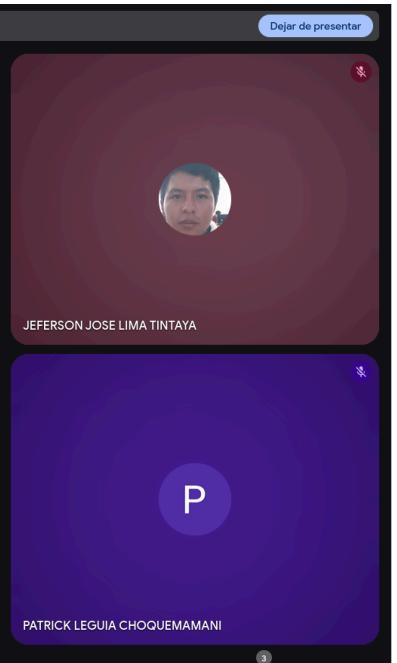
	2. Algoritmos principales	Patrick	- Elaborar pseudocódigo para: - Agregar proceso. - Cambiar estado del proceso. - Asegurar claridad y coherencia con el diseño.
	3. Diagramas de flujo	Jeferson	- Elaborar diagramas de flujo para los algoritmos. - Usar simbología estándar. - Asegurar legibilidad y consistencia visual.
	4. Justificación del diseño	Patrick	- Explicar las ventajas del diseño elegido. - Analizar eficiencia, modularidad y escalabilidad.
CAPÍTULO 3: Solución Final	1. Código limpio, comentado y estructurado	Jeferson	- Implementar código siguiendo buenas prácticas. - Comentar adecuadamente cada bloque de código. - Asegurar legibilidad y funcionalidad.
	2. Capturas de pantalla de ejecución y validación	Patrick	- Realizar pruebas de validación del sistema. - Tomar y organizar capturas de pantalla de resultados. - Documentar las pruebas exitosas y los errores corregidos.
	3. Manual de usuario	Jeferson	- Redactar el manual con instrucciones claras para el usuario final. - Incluir capturas, ejemplos y explicaciones de cada función.
CAPÍTULO 4: Evidencias de Trabajo en Equipo	1. Repositorio con control de versiones	Patrick	- Crear y mantener el repositorio colaborativo (GitHub, GitLab, etc.). - Registrar commits con mensajes claros y significativos. - Mostrar historial de ramas, fusiones y aportes individuales. - Incluir enlace al repositorio.
	2. Plan de trabajo y roles asignados	Todo el Grupo	- Redactar este documento de asignación de tareas y responsabilidades. - Elaborar cronograma con fechas de entrega parcial. - Registrar reuniones y acuerdos en actas breves.
CONTROL Y SEGUIMIENTO	Revisión y coordinación	Todo el Grupo	- Revisión conjunta del documento antes de entrega. - Reuniones semanales para seguimiento. - Entrega final en formato digital (PDF o DOCX).

2.2. Cronograma con fechas límite para cada entrega parcial.

Semana	Actividad	Capítulo	Responsables	Fecha Límite	Observaciones
Semana 1	Redacción de la Descripción del problema	Capítulo 1	Jeferson	23/10/2025	Identificar claramente el problema, objetivos y contexto.
	Definición de Requerimientos del sistema	Capítulo 1	Patrick	23/10/2025	Completar lista de requerimientos revisada por ambos.
	Propuesta de Estructuras de datos y su justificación inicial	Capítulo 1	Todo el Grupo	23/10/2025	Entregar borrador con elección de estructuras y razones técnicas.
	Descripción detallada de estructuras de datos y operaciones	Capítulo 2	Jeferson	23/10/2025	Explicar operaciones básicas y su lógica.
	Desarrollo de pseudocódigo de algoritmos principales	Capítulo 2	Patrick	23/10/2025	Crear pseudocódigo para agregar y cambiar estado del proceso.
	Elaboración de diagramas de flujo y justificación del diseño	Capítulo 2	Todo el Grupo	23/10/2025	Diagramas revisados y justificación escrita.
Semana 2	Implementar las estructuras de datos básicas	Capítulo 3	Todo el Grupo	30/10/2025	Desarrollar las funcionalidades principales del código
	Elaboración del Manual de Usuario	Capítulo 3	Jeferson	30/10/2025	Manual con instrucciones claras y fácil de entender.
Semana 3	Implementación del código final y pruebas de validación	Capítulo 3	Todo el Grupo	6/11/2025	Código funcional con capturas de pruebas correctas.
	Registro de evidencias de trabajo en equipo	Capítulo 4	Patrick	6/11/2025	Capturas del repositorio y documento de

				roles listos.
	Revisión final y entrega del informe completo	Todos los capítulos	Todo el Grupo	6/11/2025 Entregar informe final revisado y en formato PDF.

2.3. Registro de reuniones o comunicación del equipo (Actas de reuniones.).



The screenshot shows a video conference interface. On the right, there are two video feeds. The top feed shows a man with dark hair and a beard, identified as JEFERSON JOSE LIMA TINTAYA. The bottom feed shows a man with short hair, identified as PATRICK LEGUIA CHOQUEMAMANI. Both feeds have a circular profile picture next to them. To the left of the video feeds is a terminal window displaying C++ code for a client registration program. The code includes validation logic for ID and name inputs. The terminal window has a dark theme with light-colored text. At the bottom of the terminal window, there are several small icons for debugging and compilation.

```

PATRICK LEGUIA CHOQUEMAMANI (Tú, presentando) Dejar de presentar

File Edit Selection View Go Run Terminal Help
home > tom > Documentos > Estructura_de_Datos_UC > Proyecto_Automotriz_Cristo_Blanco.cpp > main()
/home/tom/Proyecto_Automotriz_Cristo_Blanco.cpp X
168     do {
169         switch (opcion) {
170             // = REGISTRAR CLIENTE CON VALIDACION ===
171             case 1:
172                 int id;
173                 string nombre, contacto;
174
175                 // ID solo numeros
176                 cout << "ID Cliente (solo numeros): ";
177                 while (!cin >> id) {
178                     if (cin.fail()) cout << "Error: Ingrese solo numeros: ";
179                     cin.clear(); // limpia el buffer para evitar errores
180                     cin.ignore(1000, '\n'); // Descarta entrada invalida
181                 }
182                 cin.ignore(); // Elimina salto de linea despues del numero
183
184                 cout << "Nombre: ";
185                 getline(cin, nombre);
186
187                 // Contacto solo numeros
188                 cout << "Contacto (solo numeros): ";
189                 while (true) {
190                     getline(cin, contacto);
191                     if (contacto.empty()) {
192                         cout << "Error: No puede estar vacio: ";
193                         continue;
194                     }
195                     bool valido = true;
196                     for (int i = 0; i < contacto.length(); i++) {
197                         if ((i < 0) || (i > 9)) {
198                             valido = false;
199                             break;
200                         }
201                     }
202                     if (valido) break;
203                     cout << "Error: Solo numeros permitidos: ";
204                 }
205
206                 registrarCliente(id, nombre, contacto);
207                 break;
208             }
209             case 2: mostrarClientes(); break;
210             case 3: int id, prioridad;
211                 string tipo;
212
213             }
214             case 4: cout << "Saliendo..." << endl;
215             exit(0);
216         }
217     }
218
219     cout << "Registros: " << registros << endl;
220
221     cout << "Presione Enter para salir..." << endl;
222     cin.get();
223
224 }
225
226
227
228

```

Line 180, Col 22 Spaces: 4 UTF-8 LF () C++ Linux