



Universidad  
Tecmilenio.

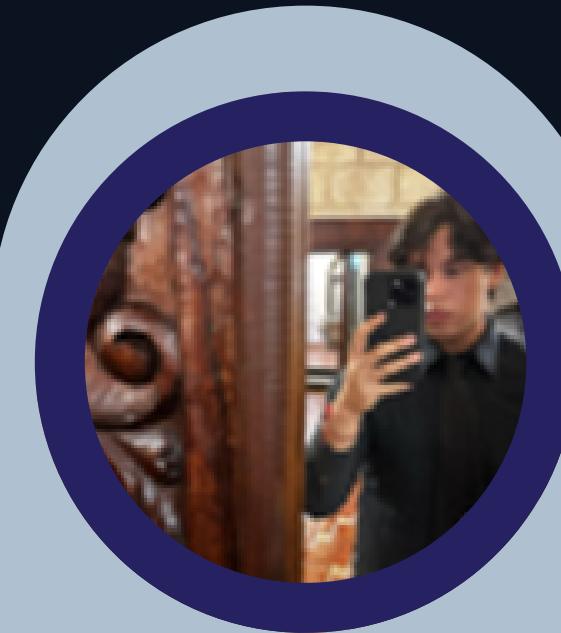
# TECH SOLUTIONS

ESTRUCTURA DE DATOS

**Presentado por el**  
**equipo 5**



# AUTORES

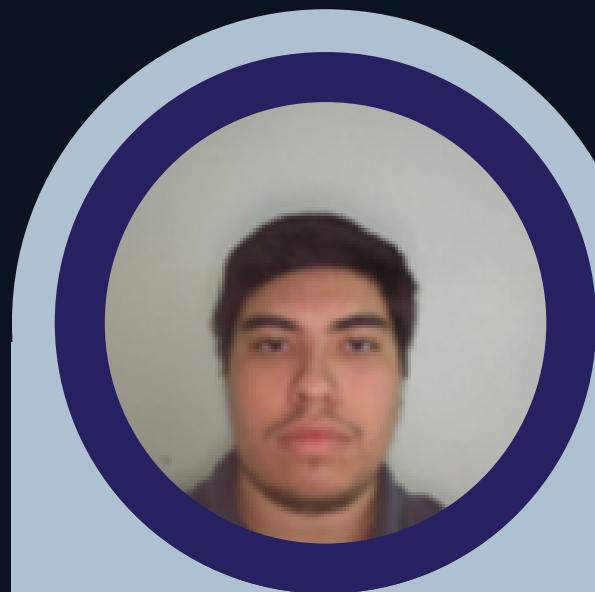


**Jesus David  
Marroquin**

**Desarollador  
tecnica**

**Mauricio  
sanchez  
figeroa**

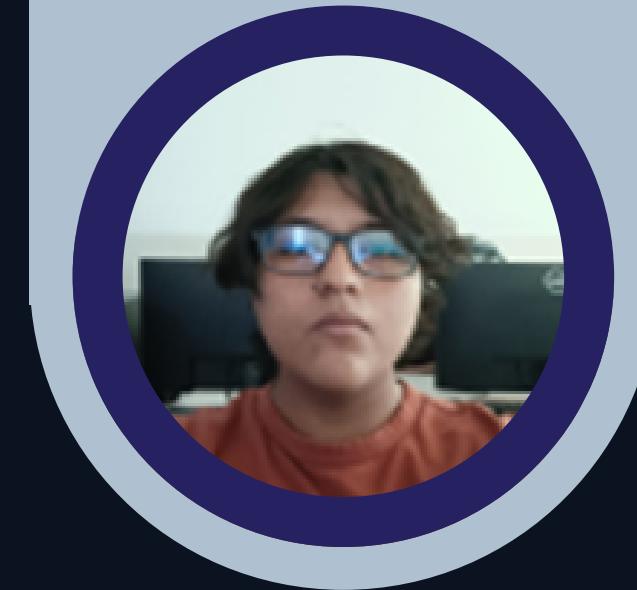
**Documentador  
analista**



**Juan Porfirio  
Torres**

**Programador  
Debugger**

**Ian Carlos  
Martinez Diaz**  
**Lider del  
equipo,  
programador**



# CASO DE ESTUDIO

La empresa TechSolutions, ubicada en Monterrey, tiene varios departamentos (Desarrollo, Marketing, Contabilidad, Recursos Humanos, etc.). Se detectaron problemas como la dispersión, falta de priorización y poca visibilidad de dependencias, lo que dio lugar a retrasos y baja productividad

Durante un diagnóstico interno se identificó que el manejo de tareas y proyectos se llevaba a cabo de manera fragmentada, principalmente mediante hojas de cálculo y listas generales sin un control centralizado. Esta situación generaba retrasos constantes, duplicidad de esfuerzos y baja productividad en los equipos de trabajo.



**TechSolutions**

# PROBLEMAS IDENTIFICADOS

## Falta de Priorización

No existía una clasificación clara entre tareas urgentes y aquellas de carácter programado, lo que ocasionaba que actividades críticas no fueran atendidas a tiempo.

## Dependencias invisible

Las relaciones entre tareas no estaban documentadas de forma estructurada, dificultando el entendimiento de qué actividades dependían de otras para su ejecución.

## Retrasos por búsqueda ineficiente

La información estaba dispersa en diferentes documentos, provocando que encontrar una tarea específica fuera un proceso lento y poco práctico.

## Escasa trazabilidad

Resultaba complicado conocer en tiempo real la carga de trabajo de cada empleado, así como el progreso de sus actividades asignadas.

# OBJETIVOS SMART

## **Específico:**

Implementar un gestor de tareas que utilice estructuras de datos especializadas (Pila LIFO, Cola FIFO, Listas, Cola de Prioridad, BST y Grafos) para priorizar tareas, visualizar dependencias y mejorar la trazabilidad del trabajo.

## **Medible:**

Asegurar que el 100% de las tareas críticas se atiendan en menos de 24 horas.

Documentar el 100% de las dependencias entre tareas.

## **Alcanzable:**

Se cuenta con el código ya desarrollado en Java 17+, MongoDB para persistencia, Swing para la interfaz, y algoritmos de ordenamiento y búsqueda implementados.

## **Relevante:**

Solucionar problemas críticos de dispersión, falta de priorización, dependencias invisibles y baja trazabilidad, mejorando la productividad y reduciendo retrasos.

## **Temporal:**

Estar completamente implementado, probado y en operación en un plazo de 5 meses.



# SOLUCIÓN PROPUESTA

Desarrollamos un programa empresarial que resuelve completamente las problemáticas identificadas mediante

- Separación de responsabilidades
- Estructuras de datos especializadas para diferentes tipos de tareas
- Algoritmos usados para búsquedas y ordenamiento
- Sistema de roles jerárquico con permisos específicos
- Persistencia en MongoDB
- Interfaz gráfica con Swing



# SOLUCIONES A FUTURO (TOBE)

## Priorizacion Automatica

Implementar un sistema unificado con algoritmos de ordenamiento por prioridad y deadlines

## Motor de busqueda avanzado

Desarrollar un buscador que use estructuras como HashMap para indexacion rapida

## Sistema de Roles y Permisos

Implementar un control de acceso basado en roles usando el patron Strategy

## Sistema de Colaboracion en Tiempo Real

Crear una plataforma colaborativa que sincronice con MongoDB

# CÓDIGO Y DIAGRAMA

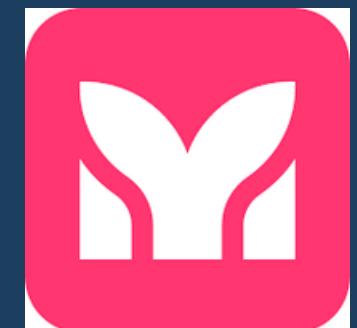


# TECNOLOGIAS USADAS

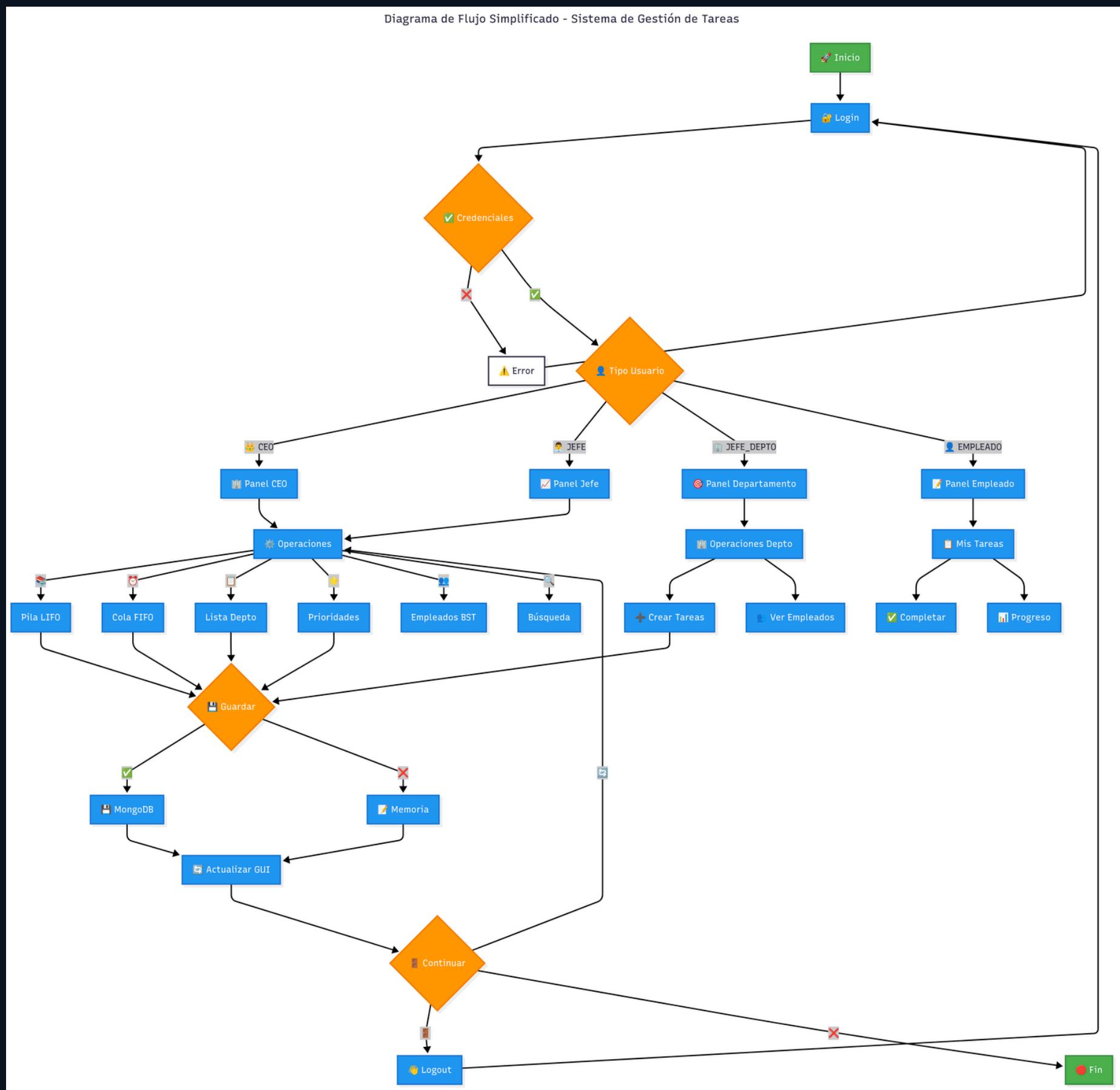
Tecnología	Version	Propósito
Java	17+	Lenguaje principal y logica de negocio
Maven	3.8+	Gestion de dependencias y construccion
MongoDB	4.0+	Base de datos para presistencia
Swing	Built-in	Interfaz grafica de usuario moderna
Mermaid	Latest	Diagramas y documentacion visual



mongoDB



# DIAGRAMA DEL PROGRAMA



# FUNCIONES PRINCIPALES DEL CODIGO

FUNCION	DESCRIPCION
ESTRUCTURA PROGRAMA	DECLARACION DE STACK, QUEUE, LIST
PILA (LIFO)	PUSH, POP, PEEK PARA TAREAS URGENTES
COLA (FIFO)	ENQUEUE, DEQUEUE, FRONT PARA TAREAS PROGRAMADAS
LISTA (ACCESO)	INSERT, DELETE, FIND TAREAS DEPARTAMENTALES
COLA PRIORIDAD	PRIORITYQUEUE CON ORDENAMIENTO AUTOMATICO
ARBOL BINARIO	BST PARA EMPLEADOS POR DEPARTAMENTO

# FUNCIONES PRINCIPALES DEL CODIGO

FUNCION	DESCRIPCION
RECUSIVIDAD	CALCULO RECUSIVO TIEMPO MAS DIVIDE Y VENCERAS
HASHMAP	ACCESO PARA BUSQUEDAS RAPIDAS
ORDENAMIENTO	QUICKSORT, MERGESORT, ALGORITMOS
BUSQUEDA	BUSQUEDA BINARIA, LINEAL, OPTIMIZADA
GRAFOS	DEPENDENCIAS, DETECCION CICLOS
INTERFAZ USUARIO	SWING CON ROLES JERARQUICOS

# ESTRUCTURA DE PROGRAMA

Definir cuatro estructuras de datos principales con referencias inmutables, integradas a MongoDB para persistencia. Incluir manejo robusto de excepciones, look-and-feel moderno automático, y soporte de roles jerárquicos con filtrado de datos por departamento del usuario autenticado.

```
94  
95  /** PILA - Para tareas urgentes (LIFO) */  
96  private final Stack<Tarea> pilaTareasUrgentes = new Stack<>();  
97  
98  /** COLA - Para tareas programadas (FIFO) */  
99  private final LinkedList<Tarea> colaTareasProgramadas = new LinkedList<>();  
100  
101 /** LISTA - Para tareas departamentales (acceso indexado) */  
102 private final List<Tarea> listaTareasDepartamento = new ArrayList<>();  
103
```

# PILA (LIFO)

Implementación completa de estructura LIFO (Last In, First Out) especializada para gestión de tareas críticas que requieren atención inmediata

```
private void agregarTareaPila() {
    try {
        Tarea tarea = mostrarDialogoNuevaTarea(tipo:"urgente");
        if (tarea != null) {
            // Validación: evitar IDs duplicados en la pila
            for (Tarea t : pilaTareasUrgentes) {
                if (t.getId().equalsIgnoreCase(tarea.getId())) {
                    JOptionPane.showMessageDialog(this, message:"Ya existe una tarea con ese ID en la pila.",
                        title:"ID duplicado", JOptionPane.WARNING_MESSAGE);
                    return;
                }
            }
            pilaTareasUrgentes.push(tarea);
            guardarTareaEnMongoDB(tarea, tipo:"urgente");
            actualizarTablaPila();
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "Error al agregar tarea: " + e.getMessage(),
            title:"Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

# COLA (FIFO)

Sistema de cola FIFO (First In, First Out)  
optimizado para tareas con secuencia  
temporal específica

```
private void agregarTareaCola() {
    try {
        Tarea tarea = mostrarDialogoNuevaTarea(tipo:"programada");
        if (tarea != null) {
            // Validación: evitar IDs duplicados en la cola
            for (Tarea t : colaTareasProgramadas) {
                if (t.getId().equalsIgnoreCase(tarea.getId())) {
                    JOptionPane.showMessageDialog(this, message:"Ya existe una tarea con ese ID en la cola.",
                        title:"ID duplicado", JOptionPane.WARNING_MESSAGE);
                    return;
                }
            }
            colaTareasProgramadas.add(tarea);
            guardarTareaEnMongoDB(tarea, tipo:"programada");
            actualizarTablaCola();
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "Error al agregar tarea: " + e.getMessage(),
            title:"Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

# LISTA (ACCESO)

Implementación de ArrayList y gestión departamental avanzada con filtrado automático por roles de usuario. Incorpora sistema de búsqueda multi-criterio (ID, departamento, urgencia, empleado asignado), operaciones de inserción/eliminación con validación de permisos

```
private void agregarTareaLista() {
    try {
        Tarea tarea = mostrarDialogoNuevaTarea(tipo:"departamento");
        if (tarea != null) {
            // Validación: evitar IDs duplicados en la lista
            for (Tarea t : listaTareasDepartamento) {
                if (t.getId().equalsIgnoreCase(tarea.getId())) {
                    JOptionPane.showMessageDialog(this, message:"Ya existe una tarea con ese ID en la lista.",
                        title:"ID duplicado", JOptionPane.WARNING_MESSAGE);
                    return;
                }
            }
            listaTareasDepartamento.add(tarea);
            guardarTareaEnMongoDB(tarea, tipo:"departamento");
            actualizarTablaLista();
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "Error al agregar tarea: " + e.getMessage(),
            title:"Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

# COLA PRIORIDAD

PriorityQueue avanzado con comparadores personalizados para ordenamiento por múltiples criterios: prioridad numérica (1=Alta, 2=Media, 3=Baja), fecha de entrega, urgencia del departamento, y tiempo estimado de completación.

```
// Nueva lógica para la tabla de prioridades
private void actualizarTablaPrioridad() {
    if (modelPrioridad == null) {
        System.err.println("Advertencia: modelPrioridad es null, omitiendo actualización");
        return;
    }
    modelPrioridad.setRowCount(rowCount:0);
}

// Copia la cola para no modificar el orden real
PriorityQueue<TareaPrioridad> copia = new PriorityQueue<>(colaPrioridad);
while (!copia.isEmpty()) {
    TareaPrioridad tarea = copia.poll();
    String prioridadStr = tarea.getPrioridad() == 1 ? "Alta" : tarea.getPrioridad() == 2 ? "Media" : "Baja";
    modelPrioridad.addRow(new Object[]{
        tarea.getId(),
        tarea.getDescripcion(),
        tarea.getDepartamento(),
        tarea.getUrgencia(),
        prioridadStr,
        tarea.getFechaEntrega()
    });
}
```

# ARBOL BINARIO

Implementación completa de BST (Binary Search Tree) auto-balanceado para gestión eficiente de empleados con búsqueda optimizada.

```
/*
 * Método recursivo para insertar empleado
 */
private Empleado insertarRec(Empleado actual, Empleado nuevo) {
    if (actual == null) {
        return nuevo;
    }

    if (nuevo.getDepartamento().compareTo(actual.getDepartamento()) < 0) {
        actual.setIzquierda(insertarRec(actual.getIzquierda(), nuevo));
    } else {
        actual.setDerecha(insertarRec(actual.getDerecha(), nuevo));
    }

    return actual;
}

/*
 * Busca empleados por departamento
 * Si departamento es null o vacío, retorna todos los empleados
 */
public void buscarPorDepartamento(String departamento, List<Empleado> resultado) {
    if (departamento == null || departamento.trim().isEmpty()) {
        buscarTodos(raiz, resultado);
    } else {
        buscarRec(raiz, departamento, resultado);
    }
}
```

# RECURSIVIDAD

Algoritmos recursivos sofisticados para cálculos complejos y optimización de recursos empresariales.

Implementa función recursiva de cola (tail recursion) para cálculo de tiempo total estimado con manejo de desbordamiento de pila, algoritmo divide y vencerás para distribución óptima de tareas entre equipos

```
// Recursividad para sumar tiempo estimado
private int calcularTiempoTotalRecursivo(List<Tarea> tareas, int idx) {
    if (idx >= tareas.size()) return 0;
    int tiempo = tareas.get(idx).getHorasEstimadas(); // Usa el campo real
    return tiempo + calcularTiempoTotalRecursivo(tareas, idx + 1);
}

// Divide y vencerás para distribuir tareas (puedes mostrar la asignación en la interfaz)
private void distribuirTareasDivideVenceras(List<Tarea> tareas, int inicio, int fin) {
    if (inicio >= fin) return;
    int medio = (inicio + fin) / 2;
    // Aquí podrías asignar tareas.get(medio) a un empleado/proyecto
    distribuirTareasDivideVenceras(tareas, inicio, medio);
    distribuirTareasDivideVenceras(tareas, medio + 1, fin);
}
```

# HASHMAP

Implementación de tabla hash optimizada para acceso O(1) constante a tareas y empleados mediante claves únicas, ademas incorpora índices secundarios para búsquedas por atributos no-clave

```
// HashMap para tareas y empleados
private final Map<String, Tarea> hashTareas = new HashMap<>();

// Metodos de ordenamiento y busqueda
private void ordenarTareasPorPrioridadFecha(List<TareaPrioridad> lista) {
    lista.sort(Comparator.naturalOrder());
}

// Ejemplo de busqueda eficiente
private Tarea buscarTareaPorId(String id) {
    return hashTareas.get(id);
}
```

# ORDENAMIENTO

algoritmos de ordenamiento optimizados para diferentes tipos de datos empresariales.

```
/*
 * Ordena tareas por prioridad usando QuickSort
 */
public static void ordenarTareasPorPrioridad(List<TareaPrioridad> tareas) {
    if (tareas == null || tareas.size() <= 1) return;
    quickSortPrioridad(tareas, bajo:0, tareas.size() - 1);
}

private static void quickSortPrioridad(List<TareaPrioridad> tareas, int bajo, int alto) {
    if (bajo < alto) {
        int pi = particionPrioridad(tareas, bajo, alto);
        quickSortPrioridad(tareas, bajo, pi - 1);
        quickSortPrioridad(tareas, pi + 1, alto);
    }
}

private static int particionPrioridad(List<TareaPrioridad> tareas, int bajo, int alto) {
    TareaPrioridad pivot = tareas.get(alto);
    int i = bajo - 1;

    for (int j = bajo; j < alto; j++) {
        if (tareas.get(j).getPrioridad() <= pivot.getPrioridad()) {
            i++;
            intercambiar(tareas, i, j);
        }
    }

    intercambiar(tareas, i + 1, alto);
    return i + 1;
}
```

# BUSQUEDA

Implementa búsqueda binaria con variantes para rangos y aproximaciones, búsqueda lineal optimizada y predicción de patrones, ademas una búsqueda interpolativa para datos uniformemente distribuidos, algoritmos de búsqueda aproximada con tolerancia configurable, y búsqueda distribuida para datasets grandes.

```
/*
 * Búsqueda binaria de tarea por ID (requiere lista ordenada por ID)
 */
public static Tarea busquedaBinariaPorId(List<Tarea> tareas, String id) {
    if (tareas == null || tareas.isEmpty() || id == null) return null;

    int izq = 0, der = tareas.size() - 1;

    while (izq <= der) {
        int medio = (izq + der) / 2;
        Tarea tareaMedio = tareas.get(medio);
        int comparacion = tareaMedio.getId().compareTo(id);

        if (comparacion == 0) {
            return tareaMedio;
        } else if (comparacion < 0) {
            izq = medio + 1;
        } else {
            der = medio - 1;
        }
    }

    return null;
}
```

# GRAFOS

Implementación completa de grafo dirigido para modelado avanzado de dependencias entre tareas y relaciones organizacionales complejas.

```
/*
 * Clase para manejar dependencias entre tareas usando un grafo
 */
public class GrafoTareas {
    private Map<String, List<String>> dependencias; // tarea -> lista de tareas de las que depende
    private Map<String, Tarea> tareas; // id -> tarea

    /**
     * Constructor
     */
    public GrafoTareas() {
        this.dependencias = new HashMap<>();
        this.tareas = new HashMap<>();
    }

    /**
     * Agrega una tarea al grafo
     */
    public void agregarTarea(Tarea tarea) {
        tareas.put(tarea.getId(), tarea);
        if (!dependencias.containsKey(tarea.getId())) {
            dependencias.put(tarea.getId(), new ArrayList<>());
        }
    }

    /**
     * Agrega una dependencia: tareaId depende de dependeDeId
     */
    public boolean agregarDependencia(String tareaId, String dependeDeId) {
        // Verificar que ambas tareas existen
        if (!tareas.containsKey(tareaId) || !tareas.containsKey(dependeDeId)) {
            return false;
        }

        // Verificar que no se cree un ciclo
        if (existeCiclo(tareaId, dependeDeId)) {
            return false;
        }

        // Agregar la dependencia
        List<String> dependenciasDeTarea = dependencias.get(tareaId);
        dependenciasDeTarea.add(dependeDeId);
    }

    /**
     * Verifica si existe un ciclo en el grafo
     */
    private boolean existeCiclo(String tareaId, String dependeDeId) {
        Set<String> visitados = new HashSet<>();
        Stack<String> pila = new Stack<>();

        pila.push(tareaId);
        visitados.add(tareaId);

        while (!pila.isEmpty()) {
            String actual = pila.pop();
            if (actual.equals(dependeDeId)) {
                return true;
            }

            List<String> dependenciasActual = dependencias.get(actual);
            if (dependenciasActual != null) {
                for (String dependencia : dependenciasActual) {
                    if (!visitados.contains(dependencia)) {
                        pila.push(dependencia);
                        visitados.add(dependencia);
                    }
                }
            }
        }

        return false;
    }
}
```

# INTERFAZ USUARIO

Sistema de interfaz gráfica empresarial desarrollado con Swing y personalización completa de Look & Feel. Implementa arquitectura de componentes reutilizables con TableWithFilters para filtrado dinámico en tiempo real, tambien un sistema de pestañas adaptativo basado en roles de usuario (JEFE: 7, JEFE\_DEPARTAMENTO: 5, EMPLEADO: 3-4),

```
public SistemaGestionTareas(Usuario usuario) {
    super(title:"Sistema de Gestion de Tareas - TechSolutions S.A. de C.V.");
    this.usuarioActual = usuario;

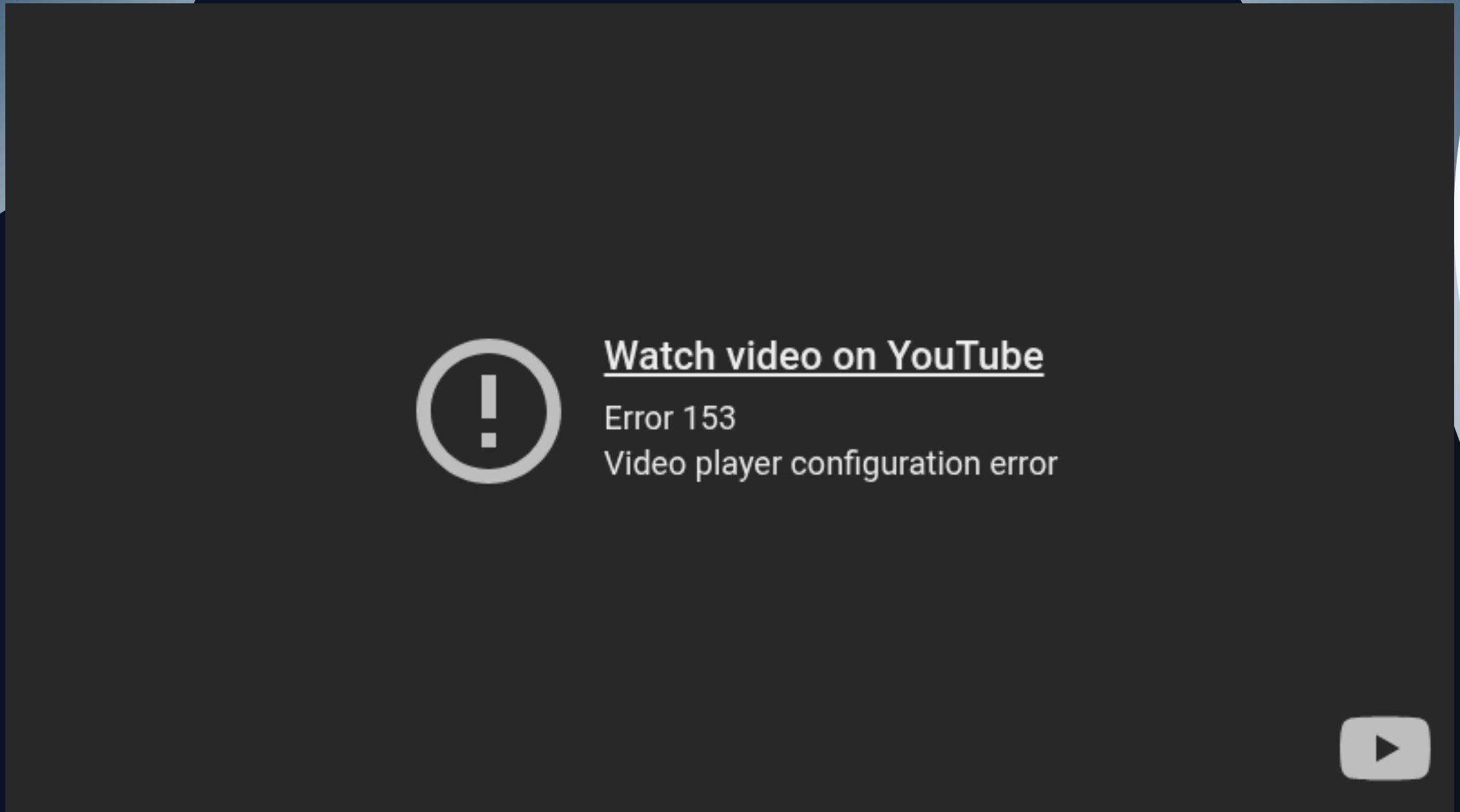
    inicializarSistema();
}

// Constructor por defecto (para compatibilidad)
public SistemaGestionTareas() {
    super(title:"Sistema de Gestion de Tareas - TechSolutions S.A. de C.V.");
    // Crear usuario temporal para testing
    this.usuarioActual = new Usuario(username:"temp", password:"temp", Usuario.Rol.JEFE, departamento:"Direccion");

    inicializarSistema();
}

/**
 * Inicializa todos los componentes del sistema
 */
private void inicializarSistema() {
    // Configurar look and feel moderno
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        // Personalizar colores y fuentes
        UIManager.put(key:"TabbedPane.selected", new Color(r:70, g:130, b:180));
        UIManager.put(key:"TabbedPane.background", new Color(r:240, g:248, b:255));
        UIManager.put(key:"Button.background", new Color(r:128, g:128, b:128));
        UIManager.put(key:"Button.foreground", Color.BLACK);
        UIManager.put(key:"Button.font", new Font(name:"Segoe UI", Font.BOLD, size:12));
        UIManager.put(key:"Table.gridColor", new Color(r:200, g:200, b:200));
        UIManager.put(key:"TableHeader.background", new Color(r:60, g:120, b:170));
        UIManager.put(key:"TableHeader.foreground", Color.BLACK);
        UIManager.put(key:"TableHeader.font", new Font(name:"Segoe UI", Font.BOLD, size:12));
    } catch (Exception e) {
        System.err.println("Error configurando Look and Feel: " + e.getMessage());
    }
}
```

# VIDEO REFLEXION



TechSolutions  
S.A. de C.V.

**GRACIAS POR  
SU ATENCION**

