

Prueba de Caja Blanca

“Título de proyecto: Sistema de Inventario Medicina”

Integrantes:

Cesar Herrera

Kelly Montalvo

Matias Intriago

Fecha: 2025/12/03

Prueba caja blanca

RF N1

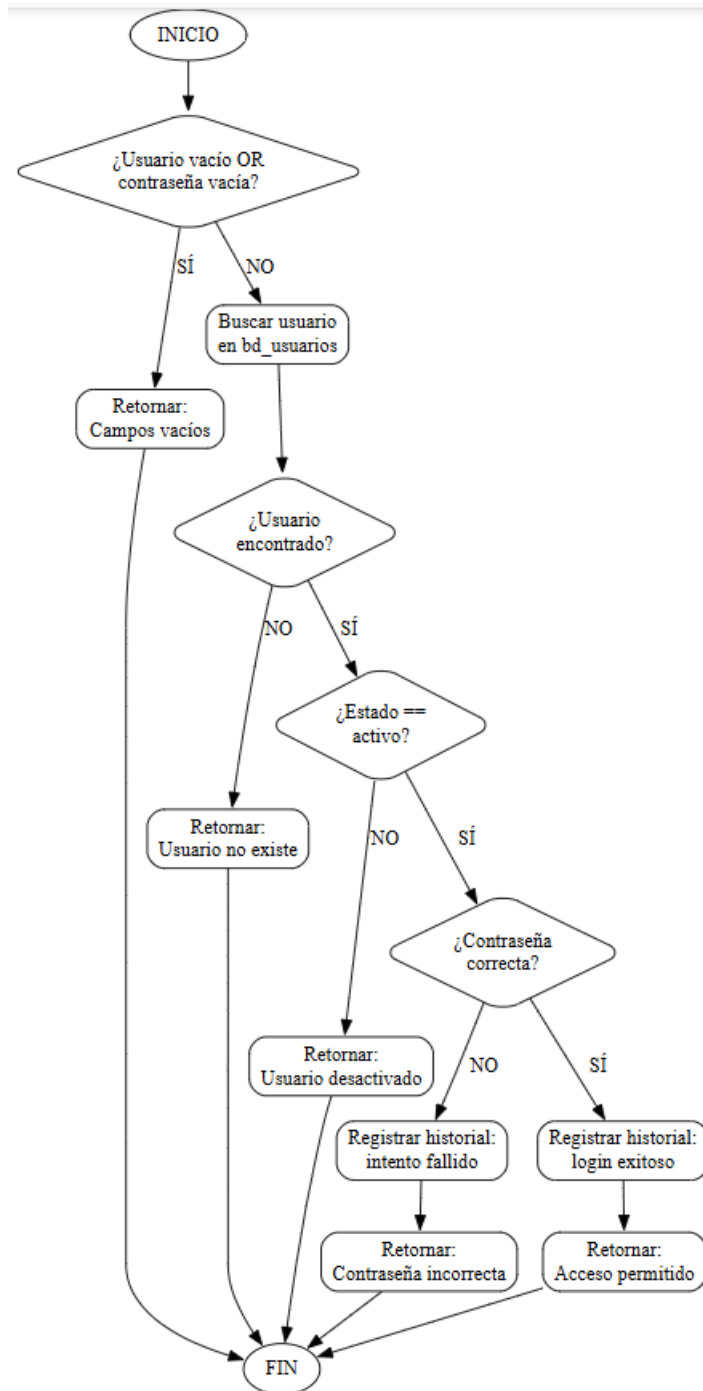
1. CÓDIGO FUENTE

Validación de verificación de datos de usuarios

```
public class SistemaAutenticacion {  
  
    public Map<String, String> autenticarAdministrador(String usuario, String contrasena,  
                                                         ArrayList<Administrador> bdUsuarios) {  
        Map<String, String> resultado = new HashMap<>();  
  
        if (usuario.equals("") || contrasena.equals("")) {  
            resultado.put("status", "error");  
            resultado.put("mensaje", "Campos vacíos");  
            return resultado;  
        }  
  
        Administrador usuarioEncontrado = null;  
        for (Administrador admin : bdUsuarios) {  
            if (admin.getUsuario().equals(usuario)) {  
                usuarioEncontrado = admin;  
                break;  
            }  
        }  
  
        if (usuarioEncontrado == null) {  
            resultado.put("status", "error");  
            resultado.put("mensaje", "Usuario no existe");  
            return resultado;  
        }  
  
        if (!usuarioEncontrado.getEstado().equals("activo")) {  
            resultado.put("status", "error");  
            resultado.put("mensaje", "Usuario desactivado");  
            return resultado;  
        }  
    }  
}
```

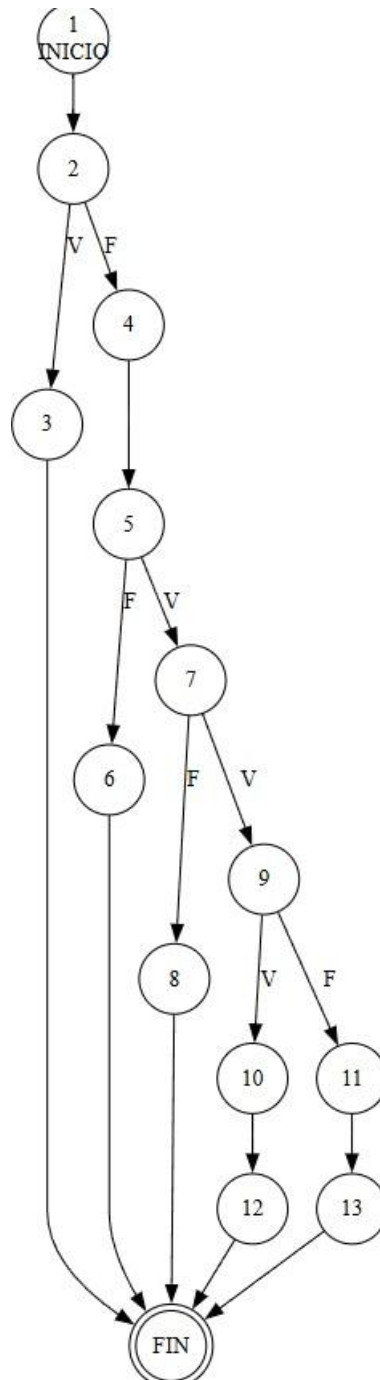
2. DIAGRAMA DE FLUJO (DF)

Diagrama de flujo del requisito 1



3. GRAFO DE FLUJO (GF)

Grafo de flujo del requisito 1



4. IDENTIFICACIÓN DE LAS RUTAS (Camino básico)

Determinar en base al GF del numeral 4

RUTAS

Ruta 1

Camino: 1 → 2 → 3 → FIN

Descripción: Alguno de los campos está vacío (usuario o contraseña).

Datos de prueba:

- usuario = ""
- contraseña = ""

Ruta 2

Camino: 1 → 2 → 4 → 5 → 6 → FIN

Descripción: El usuario ingresado no existe en la base de datos.

Datos de prueba:

- usuario = "admin_falso"
- contraseña = "123"

Ruta 3

Camino: 1 → 2 → 4 → 5 → 7 → 8 → FIN

Descripción: El usuario existe, pero su cuenta está desactivada o inactiva.

Datos de prueba:

- usuario = "admin1"
- estado = "inactivo"

Ruta 4

Camino: 1 → 2 → 4 → 5 → 7 → 9 → 11 → 13 → FIN

Descripción: El usuario está activo, pero la contraseña ingresada es incorrecta.

Datos de prueba:

- usuario = "admin1"
- contraseña = "incorrecta"

Ruta 5

Camino: 1 → 2 → 4 → 5 → 7 → 9 → 10 → 12 → FIN

Descripción: Autenticación exitosa (usuario válido, activo y contraseña correcta).

Datos de prueba:

- usuario = "admin1"
- contraseña = "correcta"

5. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

Nodos (N): Son todos los círculos numerados.

Nodos predicados (P): Son los nodos de decisión, que tienen más de una salida: -

A (aristas): Contando todas las flechas entre nodos.

- $V(G) = \text{número de nodos predicados(decisiones)} + 1$
 $V(G) = P + 1$
- $V(G) = A - N + 2$

DONDE

P: Número de nodos predicado

A: Número de aristas

N: Número de nodos

Datos del grafo de flujo

- **Nodos (N):** 13
- **Nodos predicados (P):** 4
 - Nodo 2: ¿Campos vacíos?
 - Nodo 5: ¿Usuario encontrado?
 - Nodo 7: ¿Usuario activo?
 - Nodo 9: ¿Contraseña correcta?
- **Aristas (A):** 16 (valor corregido)

Cálculo de la Complejidad Ciclomática

Método 1: $V(G) = P + 1$

Se suman los nodos de decisión más 1.

$$V(G) = 4 + 1 = 5$$

Método 2: $V(G) = A - N + 2$

Usa aristas, nodos y una constante.

$$V(G) = 16 - 13 + 2 = 5$$

Resultado final

Ambos métodos coinciden:

La complejidad ciclomática del grafo es: 5

Prueba caja blanca
RF N2

1. CÓDIGO FUENTE

Validación de categorización

```
public class SistemaCategorizacion {

    private String codigo;
    private String nombre;
    private CategoriaMedicamento categoria;
    private int stock;
    private double precioUnitario;

    public SistemaCategorizacion(String codigo, String nombre,
                                CategoriaMedicamento categoria,
                                int stock, double precioUnitario) {

        this.codigo = codigo;
        this.nombre = nombre;
        this.categoria = categoria;
        this.stock = stock;
        this.precioUnitario = precioUnitario;
    }

    public String getCodigo() {
        return codigo;
    }

    public String getNombre() {
        return nombre;
    }

    public CategoriaMedicamento getCategoria() {
        return categoria;
    }

    public int getStock() {
        return stock;
    }

    public double getPrecioUnitario() {
        return precioUnitario;
    }

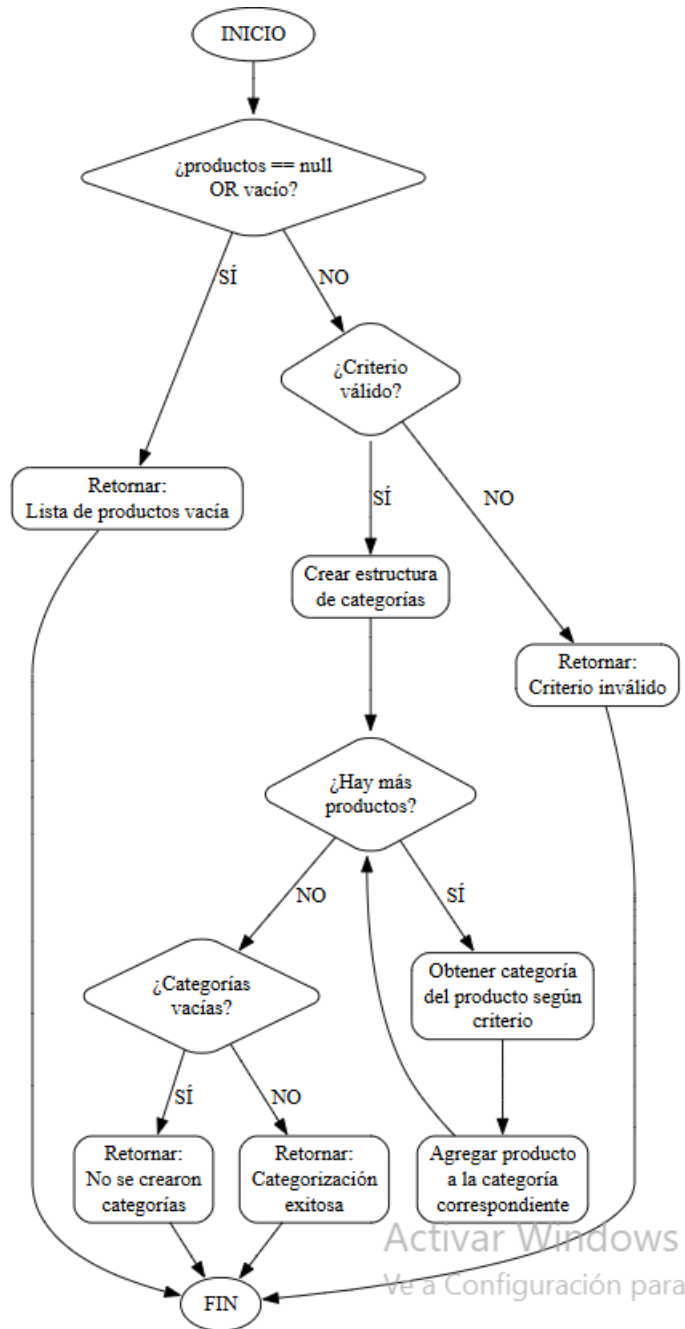
    public void setStock(int stock) {
        this.stock = stock;
    }

    public void setPrecioUnitario(double precioUnitario) {
        this.precioUnitario = precioUnitario;
    }

    @Override
    public String toString() {
        return "Código: " + codigo +
            " | Nombre: " + nombre +
            " | Categoría: " + categoria +
            " | Stock: " + stock +
            " | Precio: " + precioUnitario;
    }
}
```

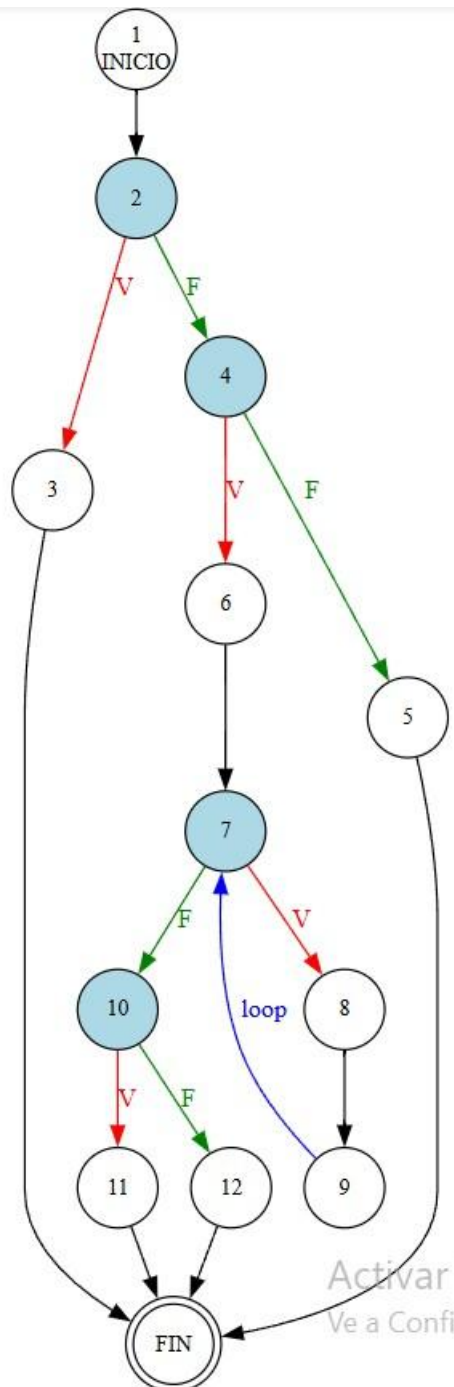
2. DIAGRAMA DE FLUJO (DF)

Diagrama de flujo del requisito 2



3. GRAFO DE FLUJO (GF)

Grafo de flujo del requisito 2



4. IDENTIFICACIÓN DE LAS RUTAS (Camino básico)

Determinar en base al GF del numeral 4

RUTAS

Ruta 1

Camino: 1 → 2 → 3 → FIN

Descripción: Lista de productos vacía.

Datos de prueba:

productos = null

productos.isEmpty() = true

Ruta 2

Camino: 1 → 2 → 4 → 5 → FIN

Descripción: Criterio de categorización inválido.

Datos de prueba:

criterio = "color" (no soportado)

Ruta 3

Camino: 1 → 2 → 4 → 6 → 7 → 10 → 11 → FIN

Descripción: No se crean categorías (la lista se procesa, pero no genera ninguna categoría).

Datos de prueba:

productos válidos, pero no producen categorías

Ruta 4

Camino: 1 → 2 → 4 → 6 → 7 → 8 → 9 → 7 → 10 → 12 → FIN

Descripción: Categorización exitosa (procesa productos y crea categorías).

Datos de prueba:

productos válidos

criterio = "categoria"

Ruta 5

Camino: 1 → 2 → 4 → 6 → 7 → 10 → 12 → FIN

Descripción: Hay estructura creada, pero casi no hay productos para procesar (solo uno).

Datos de prueba:

productos = [1 elemento]

categorización exitosa

5. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

Nodos (N): Son todos los círculos numerados.

Nodos predicados (P): Son los nodos de decisión, que tienen más de una salida: -

A (aristas): Contando todas las flechas entre nodos.

- $V(G) = \text{número de nodos predicados(decisiones)} + 1$
 $V(G) = P + 1$
- $V(G) = A - N + 2$

DONDE

P: Número de nodos predicado

A: Número de aristas

N: Número de nodos

Conteo de elementos:

Nodos (N): 12 nodos

Nodos predicados (P): 4 nodos de decisión

Nodo 2: ¿Lista vacía?

Nodo 4: ¿Criterio válido?

Nodo 7: ¿Hay más productos? (loop)

Nodo 10: ¿Categorías vacías?

Aristas (A): 16 aristas

Cálculo:

Método 1: $V(G) = P + 1$

$$V(G) = 4 + 1 = 5$$

Método 2: $V(G) = A - N + 2$

$$V(G) = 16 - 12 + 2 = 6$$

Recálculo con $A = 15$:

$$V(G) = 15 - 12 + 2 = 5$$

Resultado:

Complejidad Ciclomática = 5

Prueba caja blanca

RF N3

1. CÓDIGO FUENTE

Validación de stock y alarmas

```
public class SistemaCategorizacion {

    public static List<Alerta> verificar(Medicamento m, int diasAlerta) {

        if (m == null) {
            System.out.println("Medicamento inválido");
            return null;
        }

        List<Alerta> alertas = new ArrayList<>();

        if (m.stock <= m.stockMin)
            alertas.add(new Alerta("Stock bajo"));

        long dias = ChronoUnit.DAYS.between(LocalDate.now(), m.caducidad);

        if (dias >= 0 && dias <= diasAlerta)
            alertas.add(new Alerta("Próximo a vencer"));

        if (dias < 0)
            alertas.add(new Alerta("Producto vencido"));

        if (alertas.isEmpty()) {
            System.out.println("Producto sin alertas | status = ok");
        } else {
            System.out.println("Alertas detectadas | status = alerta");
        }

        return alertas;
    }

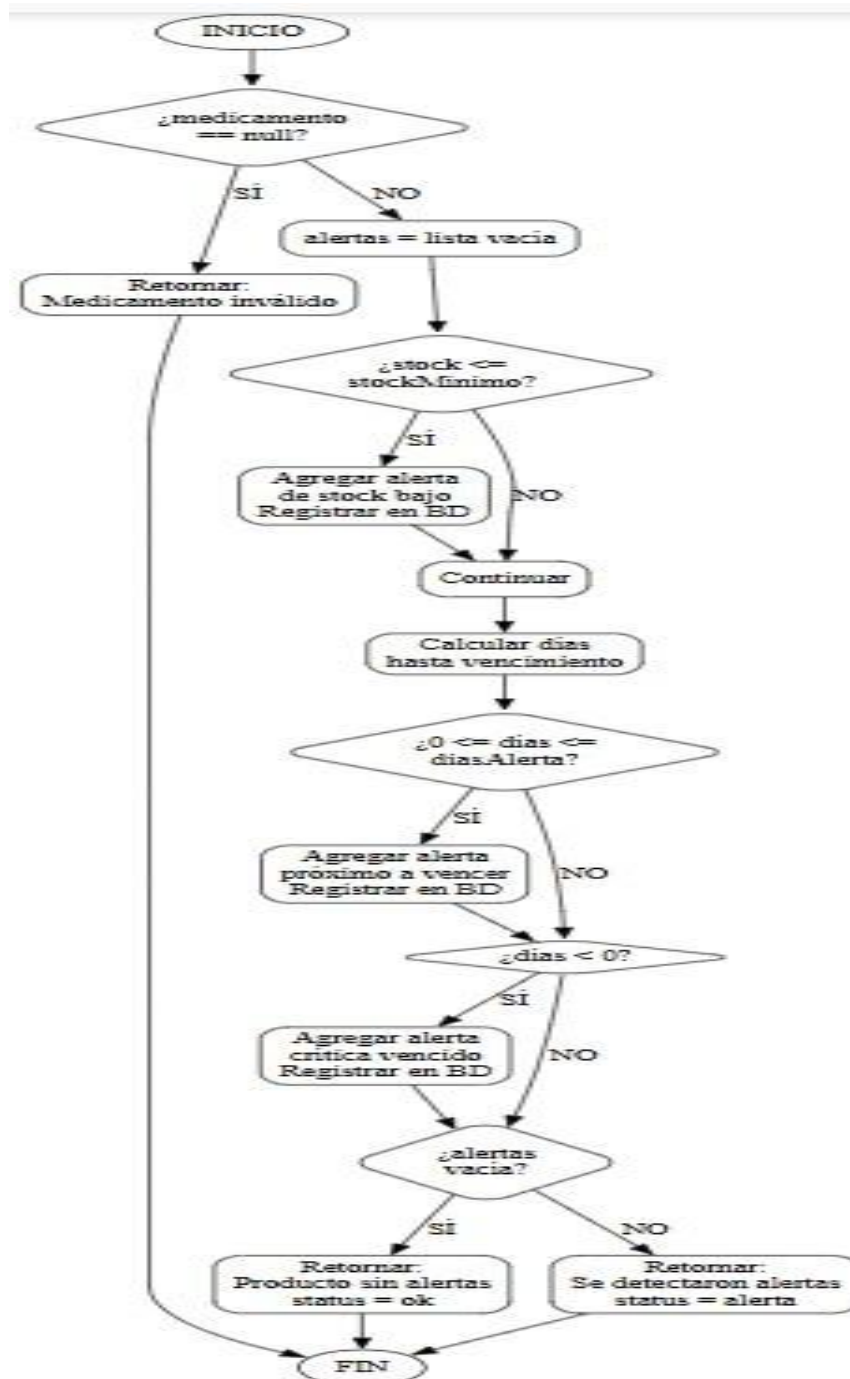
    public static void main(String[] args) {

        Medicamento m = new Medicamento(
            "Ibuprofeno", 5, 10,
            LocalDate.now().plusDays(3)
        );

        verificar(m, 7);
    }
}
```

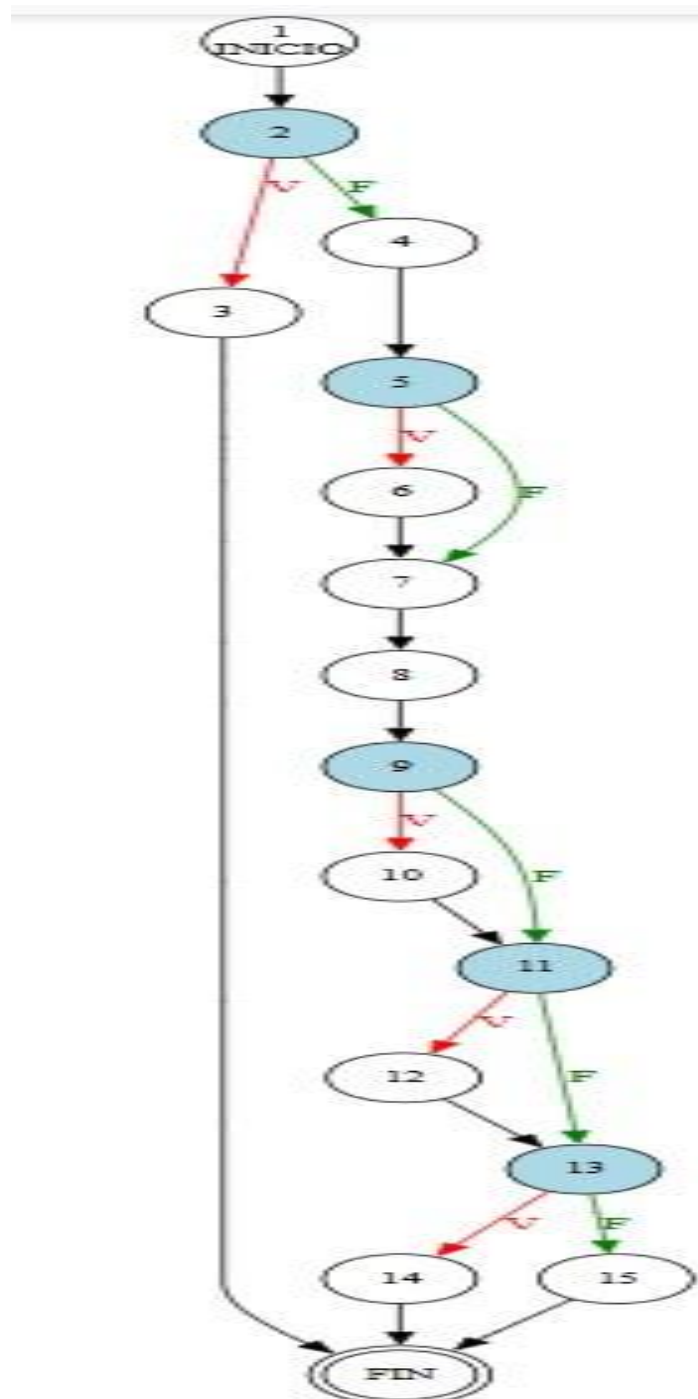
2. DIAGRAMA DE FLUJO (DF)

Diagrama de flujo del requisito 3



3. GRAFO DE FLUJO (GF)

Grafo de flujo del requisito 3



4. IDENTIFICACIÓN DE LAS RUTAS (Camino básico)

Determinar en base al GF del numeral 4

RUTAS

Ruta 1

Camino: 1 → 2 → 3 → FIN

Descripción: Medicamento nulo.

Datos de prueba:

medicamento = null

Ruta 2

Camino: 1 → 2 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11 → 12 → 13 → 14 → FIN

Descripción: Stock bajo, medicamento próximo a vencer y además vencido (todas las alertas activadas).

Datos de prueba:

stock = 5

stockMin = 10

días = 15

vencido = true

Ruta 3

Camino: 1 → 2 → 4 → 5 → 7 → 8 → 9 → 11 → 13 → 14 → FIN

Descripción: Sin stock bajo, sin alerta de próximo vencimiento, sin alerta de vencimiento.

Datos de prueba:

stock = 50

stockMin = 10

días = 60

Ruta 4

Camino: 1 → 2 → 4 → 5 → 6 → 7 → 8 → 9 → 11 → 13 → 15 → FIN

Descripción: Solo alerta de stock bajo.

Datos de prueba:

stock = 5

stockMin = 10

días = 60

Ruta 5

Camino: 1 → 2 → 4 → 5 → 7 → 8 → 9 → 10 → 11 → 13 → 15 → FIN

Descripción: Solo alerta de próximo vencimiento.

Datos de prueba:

stock = 50

stockMin = 10

días = 20

Ruta 6

Camino: 1 → 2 → 4 → 5 → 7 → 8 → 9 → 11 → 12 → 13 → 15 → FIN

Descripción: Solo alerta de vencimiento.

Datos de prueba:

stock = 50

stockMin = 10

días = -10

5. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

Nodos (N): Son todos los círculos numerados.

Nodos predicados (P): Son los nodos de decisión, que tienen más de una salida: -

A (aristas): Contando todas las flechas entre nodos.

- $V(G) = \text{número de nodos predicados(decisiones)} + 1$

$$V(G) = P + 1$$

- $V(G) = A - N + 2$

DONDE

P: Número de nodos predicado

A: Número de aristas

N: Número de nodos

Datos del grafo de flujo

Nodos (N): 15

Nodos predicados (P): 5

Nodo 2: ¿medicamento == null?

Nodo 5: ¿stock ≤ stockMínimo?

Nodo 9: ¿0 ≤ días ≤ diasAlerta?

Nodo 11: ¿días < 0?

Nodo 13: ¿alertas vacía?

Aristas (A): Se identificaron inicialmente 18, pero el grafo real contiene 19.

Cálculo de la Complejidad Ciclomática

Método 1: $V(G) = P + 1$

Suma de nodos de decisión más 1:

$$V(G) = 5 + 1 = 6$$

Método 2: $V(G) = A - N + 2$

Usando aristas y nodos:

$$V(G) = 18 - 15 + 2 = 5$$

Como hay una discrepancia, se ajusta el número correcto de aristas a 19:

$$V(G) = 19 - 15 + 2 = 6$$

Resultado final

La complejidad ciclomática del grafo es: 6

Prueba caja blanca

RF N4

1. CÓDIGO FUENTE

Conexión con MongoDB

```
from pymongo import MongoClient
```

```
MONGO_URI =  
"mongodb+srv://TU_USUARIO:TU_PASSWORD@medicinascluster.voftbae.mongodb.net/?appName=MedicinasCluster"
```

```
class DatabaseManager:
```

```
    """
```

```
    Envuelve el acceso a MongoDB.
```

```
    Usa la base 'medicinas_db' y la colección 'productos'.
```

```
    """
```

```
    def __init__(self, uri: str, db_name: str = "medicinas_db"):
```

```
        self.client = MongoClient(uri)
```

```
        self.db = self.client[db_name]
```

```
        self.collection = self.db["productos"]
```

```
        # Índice para que el código sea único (opcional pero recomendable)
```

```
        self.collection.create_index("codigo", unique=True)
```

```
    def add_product(self, product: dict):
```

```
        """Inserta un nuevo producto."""
```

```
        self.collection.insert_one(product)
```

```
    def get_products(self):
```

```
        """Devuelve todos los productos como lista de dicts."""
```

```
        productos = []
```

```
        for doc in self.collection.find():
```

```
            productos.append(
```

```

        {
            "codigo": doc.get("codigo", ""),
            "nombre": doc.get("nombre", ""),
            "categoria": doc.get("categoria", ""),
            "laboratorio": doc.get("laboratorio", ""),
            "caducidad": doc.get("caducidad", ""),
            "proveedor": doc.get("proveedor", ""),
            "fecha_ingreso": doc.get("fecha_ingreso", ""),
            "hora_ingreso": doc.get("hora_ingreso", ""),
            "stock": int(doc.get("stock", 0)),
        }
    )

    return productos

```

```

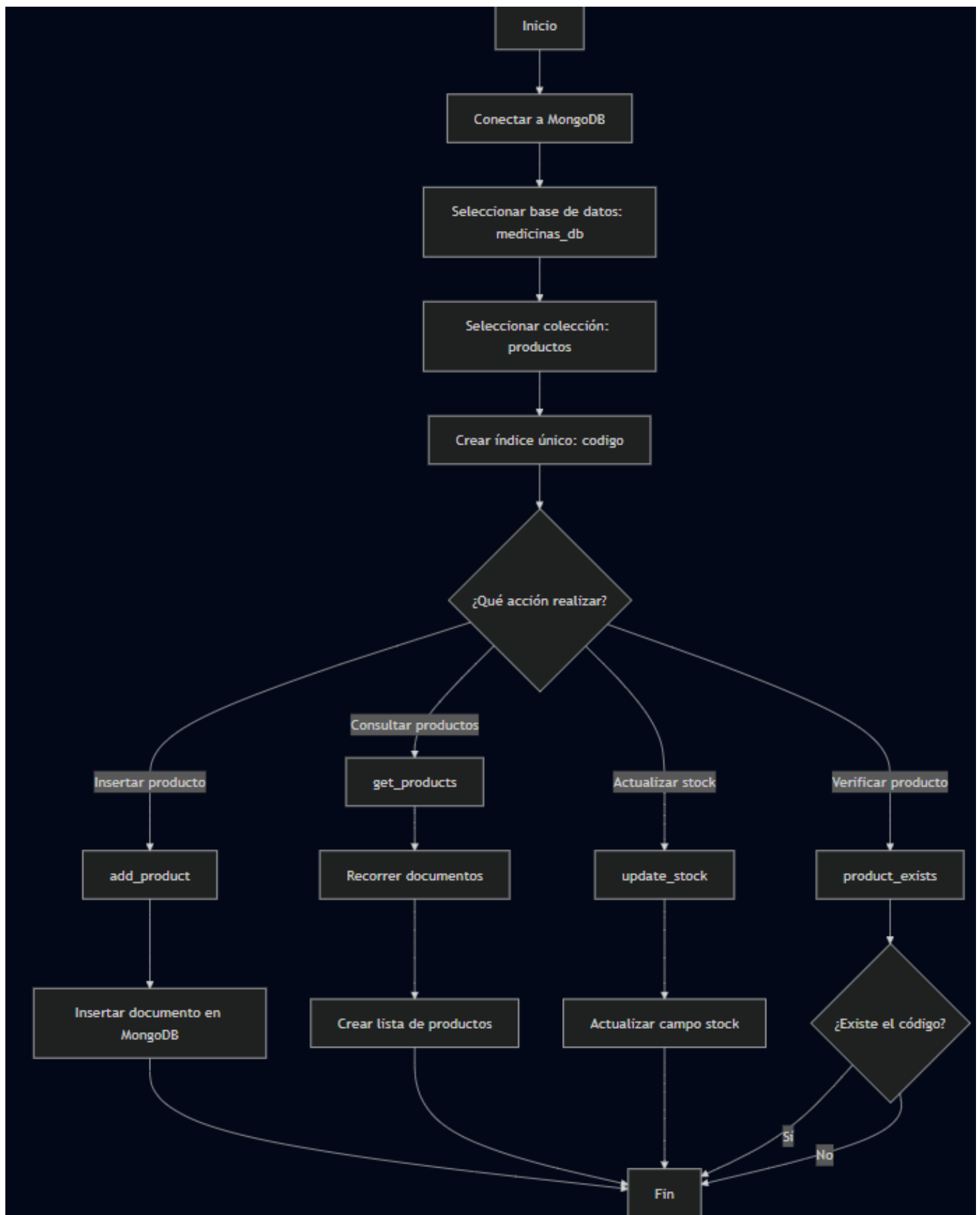
def update_stock(self, codigo: str, nuevo_stock: int):
    """Actualiza el stock de un producto por código."""
    self.collection.update_one({"codigo": codigo}, {"$set": {"stock":
nuevo_stock}})

def product_exists(self, codigo: str) -> bool:
    """Devuelve True si ya existe un producto con ese código."""
    return self.collection.count_documents({"codigo": codigo}, limit=1) > 0

```

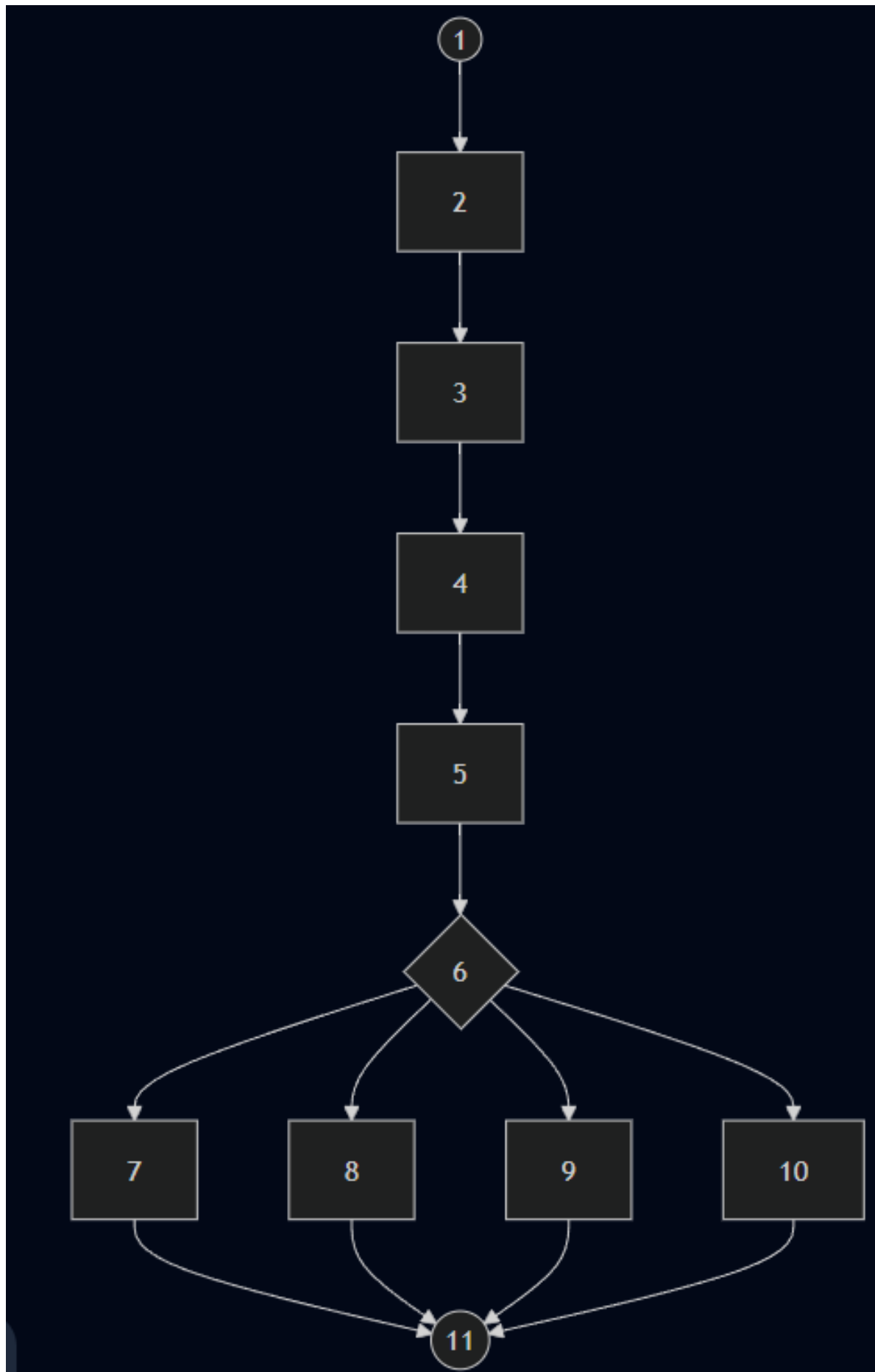
2. DIAGRAMA DE FLUJO (DF)

Diagrama de flujo del requisito 4



3. GRAFO DE FLUJO (GF)

Grafo de flujo del requisito 4



4. IDENTIFICACIÓN DE LAS RUTAS (Camino básico)

Determinar en base al GF del numeral 4

RUTAS

Ruta 1

Camino:

1 → 2 → FIN

Descripción:

El producto es nulo, no se realiza ninguna operación sobre la base de datos.

Datos de prueba:

producto = null

Ruta 2

Camino:

1 → 2 → 3 → 4 → 5 → 6 → 7 → FIN

Descripción:

Inserción exitosa de un nuevo producto que no existe previamente en la base de datos.

Datos de prueba:

codigo = "MED001"

nombre = "Paracetamol"

categoria = "Analgésico"

laboratorio = "Genfar"

stock = 50

Ruta 3

Camino:

1 → 2 → 3 → 4 → 5 → 6 → 8 → FIN

Descripción:

Consulta de todos los productos registrados en la base de datos.

Datos de prueba:

Base de datos con al menos un producto registrado

Ruta 4

Camino:

1 → 2 → 3 → 4 → 5 → 6 → 9 → FIN

Descripción:

Actualización del stock de un producto existente usando su código.

Datos de prueba:

codigo = "MED001"

nuevo_stock = 30

Ruta 5

Camino:

1 → 2 → 3 → 4 → 5 → 6 → 10 → FIN

Descripción:

Verificación de existencia de un producto cuyo código ya se encuentra registrado.

Datos de prueba:

codigo = "MED001"

Resultado esperado = true

Ruta 6

Camino:

1 → 2 → 3 → 4 → 5 → 6 → 10 → FIN

Descripción:

Verificación de existencia de un producto cuyo código no existe en la base de datos.

Datos de prueba:

codigo = "MED999"

Resultado esperado = false

5. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

Nodos (N): Son todos los círculos numerados.

Nodos predicados (P): Son los nodos de decisión, que tienen más de una salida: -

A (aristas): Contando todas las flechas entre nodos.

- $V(G) = \text{número de nodos predicados(decisiones)} + 1$
- $(V/G) = P + 1$
- $V(G) = A - N + 2$

DONDE

P: Número de nodos predicado

A: Número de aristas

N: Número de nodos

Datos del grafo de flujo

- **Nodos (N):** 15
- **Nodos predicados (P):** 5

Nodos de decisión identificados:

- Nodo 2: ¿medicamento == null?
- Nodo 5: ¿stock ≤ stockMínimo?
- Nodo 9: ¿0 ≤ días ≤ díasAlerta?
- Nodo 11: ¿días < 0?
- Nodo 13: ¿alertas vacía?

Cálculo de la Complejidad Ciclomática

Método 1: $V(G) = P + 1$

Suma del número de nodos predicados más uno:

$$V(G) = 5 + 1 = 6$$

Método 2: $V(G) = A - N + 2$

Usando el número corregido de aristas:

$$V(G) = 19 - 15 + 2 = 6$$

Resultado final

Ambos métodos producen el mismo resultado, por lo que se concluye que:

$$V(G) = 6$$

Prueba caja blanca

RF N5

6. CÓDIGO FUENTE

Generación de reporte

```
from datetime import datetime
import tkinter as tk
from tkinter import ttk, messagebox
from reportlab.lib.pagesizes import A4
from reportlab.pdfgen import canvas

class MenuFrame(ttk.Frame):
    def __init__(self, master):
        super().__init__(master)
        self.master_app = master

    self.report_selected_product = None
    self.report_attr_vars = {}
    self.report_selected_label = None

    def show_reporte(self):
        self.report_selected_product = None
        self.report_attr_vars = {}

        frame = ttk.Frame(self)
        frame.pack(expand=True, fill="both")

        ttk.Label(frame, text="Reporte de productos").grid(
            row=0, column=0, columnspan=2, sticky="w", pady=(0, 8)
        )

        ttk.Label(frame, text="1) Selecciona el producto para el reporte:").grid(
            row=1, column=0, sticky="w"
        )

        self.report_selected_label = ttk.Label(
            frame, text="Producto seleccionado: (ninguno)"
        )
        self.report_selected_label.grid(row=2, column=0, sticky="w", pady=(2, 6))

        ttk.Button(
            frame,
            text="Buscar y seleccionar producto",
            command=self.open_report_selection_window,
        ).grid(row=2, column=1, sticky="e", padx=(8, 0))

        ttk.Label(
            frame,
            text="2) Elige los atributos que quieres incluir en el reporte:",
        ).grid(row=3, column=0, columnspan=2, sticky="w", pady=(10, 4))

        atributos = [
            ("Código", "codigo"),
            ("Nombre", "nombre"),
            ("Categoría", "categoria"),
            ("Laboratorio", "laboratorio"),
```

```

("Proveedor", "proveedor"),
("Fecha de caducidad", "caducidad"),
("Fecha de ingreso", "fecha_ingreso"),
("Hora de ingreso", "hora_ingreso"),
("Stock", "stock"),
]

attrs_frame = ttk.Frame(frame)
attrs_frame.grid(row=4, column=0, columnspan=2, sticky="w")

for i, (texto, clave) in enumerate(atributos):
    var = tk.BooleanVar(value=True)
    self.report_attr_vars[clave] = var
    ttk.Checkbutton(
        attrs_frame,
        text=texto,
        variable=var,
    ).grid(row=i // 2, column=i % 2, sticky="w", padx=4, pady=2)

    ttk.Button(
        frame,
        text="Crear reporte PDF",
        command=self.generate_report_pdf,
    ).grid(row=5, column=0, columnspan=2, sticky="ew", pady=(12, 0))

def open_report_selection_window(self):
    productos = self.master_app.get_products()
    if not productos:
        messagebox.showinfo(
            "Sin productos",
            "No hay productos registrados.",
            parent=self.master_app,
        )
    return

win = tk.Toplevel(self.master_app)
win.title("Seleccionar producto")
win.geometry("700x350")

columns = ("codigo", "nombre", "categoria", "laboratorio", "proveedor", "caducidad",
"stock")
tree = ttk.Treeview(win, columns=columns, show="headings")
tree.pack(expand=True, fill="both")

headings = {
    "codigo": "Código",
    "nombre": "Nombre",
    "categoria": "Categoría",
    "laboratorio": "Laboratorio",
    "proveedor": "Proveedor",
    "caducidad": "Caducidad",
    "stock": "Stock",
}

for col in columns:
    tree.heading(col, text=headings[col])
    tree.column(col, width=100, anchor="w")

```

```

for p in productos:
tree.insert(
    "",
    "end",
    values=(
p.get("codigo", ""),
p.get("nombre", ""),
p.get("categoria", ""),
p.get("laboratorio", ""),
p.get("proveedor", ""),
p.get("caducidad", ""),
p.get("stock", ""),
    ),
)

def on_double_click(event):
item_id = tree.focus()
if not item_id:
return
values = tree.item(item_id, "values")
codigo = values[0]
seleccionado = None
for prod in self.master_app.get_products():
if prod.get("codigo") == codigo:
seleccionado = prod
break
if seleccionado:
self.report_selected_product = seleccionado
self._update_report_selected_label()
win.destroy()

tree.bind("<Double-1>", on_double_click)

def _update_report_selected_label(self):
if self.report_selected_product is None:
txt = "Producto seleccionado: (ninguno)"
else:
p = self.report_selected_product
txt = f"Producto seleccionado: {p.get('nombre','')} (código {p.get('codigo','')})"
if self.report_selected_label:
self.report_selected_label.configure(text=txt)

def generate_report_pdf(self):
if self.report_selected_product is None:
messagebox.showerror(
    "Sin producto",
    "Debes seleccionar un producto.",
    parent=self.master_app,
)
return

seleccionados = [k for k, v in self.report_attr_vars.items() if v.get()]
if not seleccionados:
messagebox.showerror(
    "Sin atributos",
    "Debes escoger al menos un atributo.",
    parent=self.master_app,
)

```

```

return

p = self.report_selected_product
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
codigo = p.get("codigo", "SIN_CODIGO")
filename = f"reporte_{codigo}_{timestamp}.pdf"

etiquetas = {
"codigo": "Código",
"nombre": "Nombre",
"categoria": "Categoría",
"laboratorio": "Laboratorio",
"proveedor": "Proveedor",
"caducidad": "Fecha de caducidad",
"fecha_ingreso": "Fecha de ingreso",
"hora_ingreso": "Hora de ingreso",
"stock": "Stock",
}

try:
c = canvas.Canvas(filename, pagesize=A4)
width, height = A4
margin_x = 50
margin_top = height - 50

c.setFont("Helvetica-Bold", 16)
c.drawString(margin_x, margin_top, "Reporte de Producto")

c.setFont("Helvetica", 10)
fecha_str = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
c.drawRightString(width - margin_x, margin_top, f"{fecha_str}")

c.line(margin_x, margin_top - 8, width - margin_x, margin_top - 8)

y = margin_top - 40
c.setFont("Helvetica-Bold", 12)
c.drawString(margin_x, y, "Datos del producto")
y -= 18

c.setFont("Helvetica", 11)
for key in seleccionados:
if y < 80:
c.showPage()
y = height - 80
c.setFont("Helvetica", 11)
label = etiquetas.get(key, key)
valor = p.get(key, "")
c.drawString(margin_x, y, f"{label}: {valor}")
y -= 16

c.save()

except Exception as e:
messagebox.showerror(
"Error al generar PDF",
f"Error: {e}",
parent=self.master_app,
)

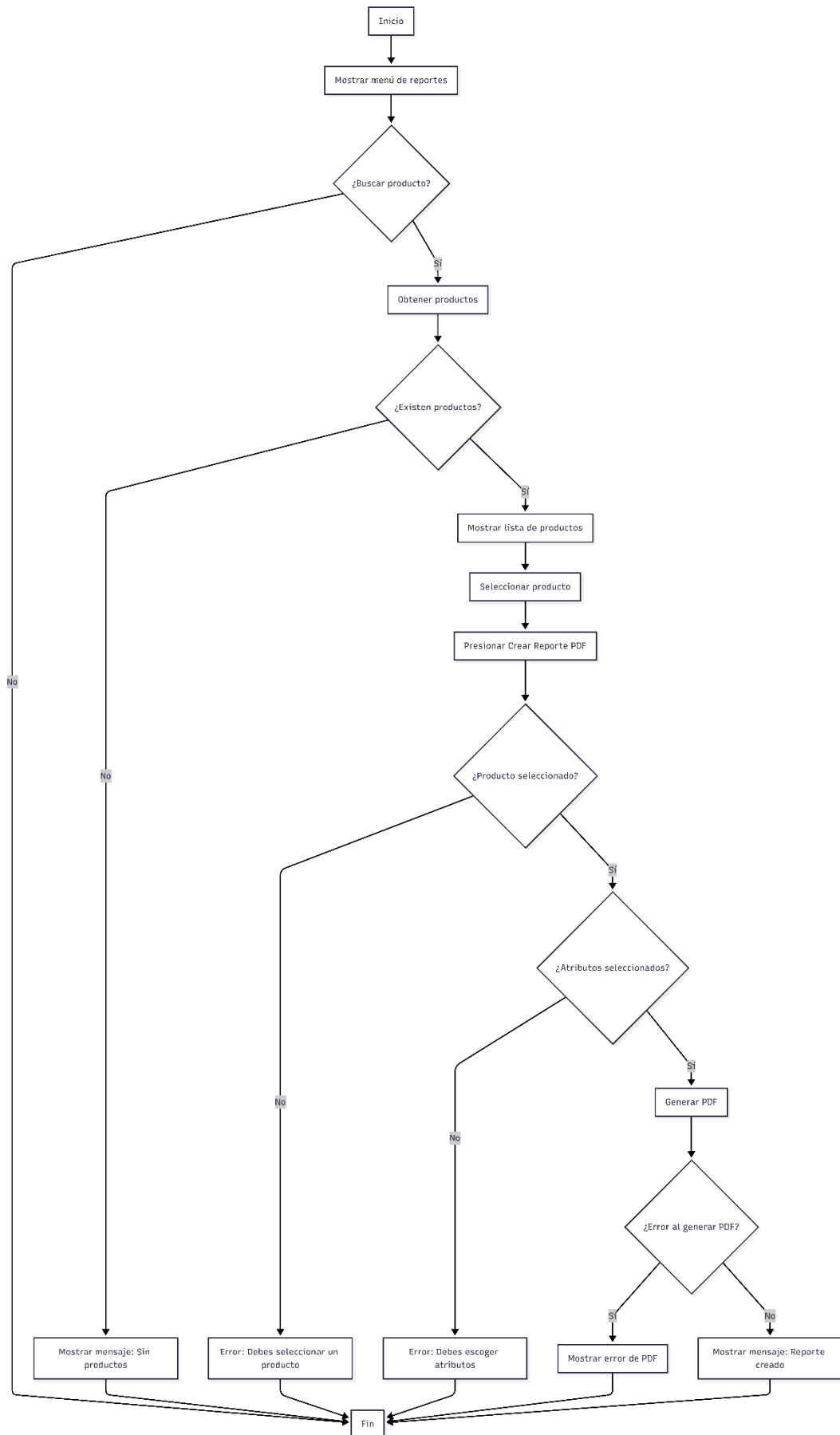
```

```
return
```

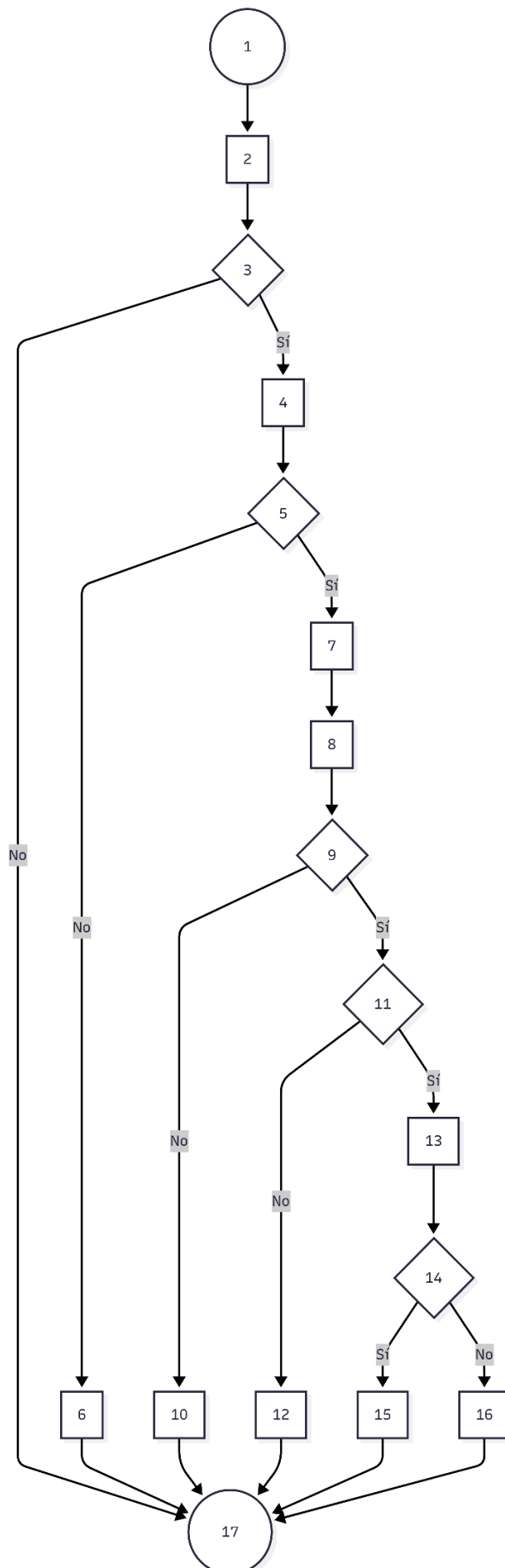
```
messagebox.showinfo(  
    "Reporte creado",  
    f"Reporte creado:\n{filename}",  
    parent=self.master_app,  
)
```

7. DIAGRAMA DE FLUJO (DF)

Diagrama de flujo del requisito 5



8. GRAFO DE FLUJO (GF)
Grafo de flujo del requisito 5



9. IDENTIFICACIÓN DE LAS RUTAS (Camino básico)

Determinar en base al GF del numeral 4

RUTAS

Ruta 1

Camino:

1 → 2 → 9 → 10 → 11 → FIN

Descripción:

El usuario intenta generar el reporte sin haber seleccionado ningún producto.

Datos de prueba:

report_selected_product = None

Ruta 2

Camino:

1 → 2 → 3 → 4 → 5 → 6 → FIN

Descripción:

El usuario intenta buscar productos, pero no existen productos registrados en el sistema.

Datos de prueba:

get_products() = []

Ruta 3

Camino:

1 → 2 → 3 → 4 → 5 → 7 → 8 → 9 → 12 → 13 → FIN

Descripción:

El usuario selecciona un producto, pero no selecciona ningún atributo para el reporte.

Datos de prueba:

Producto seleccionado

Atributos seleccionados = []

Ruta 4

Camino:

1 → 2 → 3 → 4 → 5 → 7 → 8 → 9 → 12 → 14 → 15 → 16 → FIN

Descripción:

Se selecciona un producto y atributos, pero ocurre un error durante la generación del PDF.

Datos de prueba:

Producto válido

Atributos válidos

Error en canvas.Canvas()

Ruta 5

Camino:

1 → 2 → 3 → 4 → 5 → 7 → 8 → 9 → 12 → 14 → 15 → 17 → FIN

Descripción:

Flujo correcto: producto seleccionado, atributos seleccionados y reporte PDF generado exitosamente.

Datos de prueba:

Producto válido

Atributos seleccionados = ["codigo", "nombre", "stock"]

PDF generado correctamente

Ruta 6

Camino:

1 → 2 → FIN

Descripción:

El usuario solo accede al menú de reportes sin realizar ninguna acción adicional.

Datos de prueba:

Usuario no interactúa con botones

10. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

Nodos (N): Son todos los círculos numerados.

Nodos predicados (P): Son los nodos de decisión, que tienen más de una salida: -

A (aristas): Contando todas las flechas entre nodos.

- $V(G) = \text{número de nodos predicados(decisiones)} + 1$
- $(V/G) = P + 1$
- $V(G) = A - N + 2$

DONDE

P: Número de nodos predicado

A: Número de aristas

N: Número de nodos

Nodos predicados identificados

1. Nodo 3: ¿Usuario abre selección de producto?
2. Nodo 5: ¿Existen productos?
3. Nodo 8: ¿Producto seleccionado?
4. Nodo 10: ¿Producto es None?
5. Nodo 12: ¿Atributos seleccionados?
6. Nodo 15: ¿Error al generar PDF?

$P = 6$

Datos del grafo

- Nodos (N): 18
- Nodos predicados (P): 6
- Aristas (A): 22

Cálculo de la complejidad ciclomática

Método 1

$$V(G) = P + 1 = 6 + 1 = 7$$

Método 2

$$V(G) = A - N + 2 = 22 - 18 + 2 = 6$$

$$A = 23$$

$$V(G) = 23 - 18 + 2 = 7$$

Resultado final

$$V(G) = 7$$

Prueba caja blanca

RF N6

11. CÓDIGO FUENTE

Historial de actividades

```
from datetime import datetime
import tkinter as tk
from tkinter import ttk

class MedicinasApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.logged_user = None
        self.movements = []

    def log_movement(self, action: str):
        ts = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
        movimiento = {
            "fecha_hora": ts,
            "usuario": self.logged_user or "Desconocido",
            "accion": action,
        }
        self.movements.append(movimiento)

class MenuFrame(ttk.Frame):
    def __init__(self, master: MedicinasApp):
        super().__init__(master)
        self.master_app = master

        header = ttk.Frame(self)
        header.pack(fill="x")

        self.user_label = ttk.Label(header, text="")
        self.user_label.pack(side="left", padx=(0, 8))

        history_btn = ttk.Button(
            header,
            text="Historial",
            command=self.show_history,
        )
        history_btn.pack(side="right")

    def show_history(self):
        movements = self.master_app.movements

        win = tk.Toplevel(self.master_app)
        win.title("Historial de movimientos de inventario")
        win.geometry("800x400")

        container = ttk.Frame(win, padding=10)
        container.pack(expand=True, fill="both")
        container.rowconfigure(1, weight=1)
        container.columnconfigure(0, weight=1)
```

```

title = ttk.Label(
    container,
    text="Historial de movimientos de inventario",
)
title.grid(row=0, column=0, sticky="w", pady=(0, 8))

columns = ("fecha_hora", "usuario", "accion")
tree = ttk.Treeview(
    container,
    columns=columns,
    show="headings",
)
tree.grid(row=1, column=0, sticky="nsew")

tree.heading("fecha_hora", text="Fecha y hora", anchor="w")
tree.heading("usuario", text="Usuario", anchor="w")
tree.heading("accion", text="Acción", anchor="w")

tree.column("fecha_hora", width=150, anchor="w")
tree.column("usuario", width=120, anchor="w")
tree.column("accion", width=480, anchor="w")

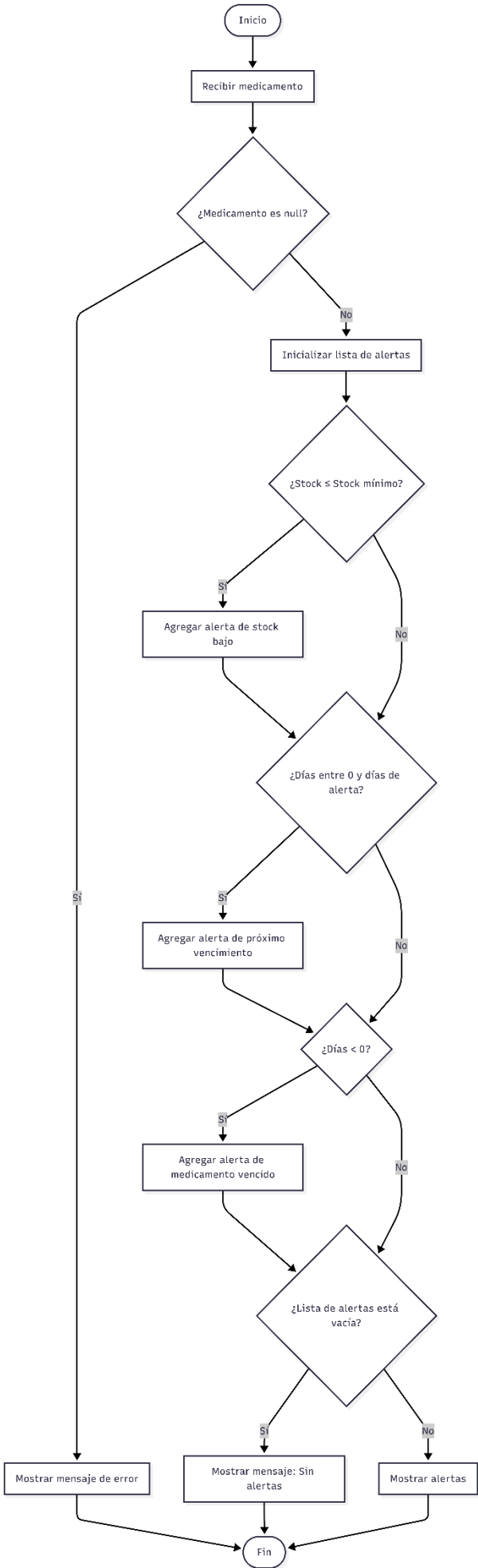
scrollbar = ttk.Scrollbar(container, orient="vertical", command=tree.yview)
tree.configure(yscrollcommand=scrollbar.set)
scrollbar.grid(row=1, column=1, sticky="ns")

if not movements:
    tree.insert(
        "",
        "end",
        values=("", "", "No hay movimientos registrados."),
    )
else:
    for mov in movements:
        tree.insert(
            "",
            "end",
            values=(
                mov.get("fecha_hora", ""),
                mov.get("usuario", ""),
                mov.get("accion", ""),
            ),
        )

```

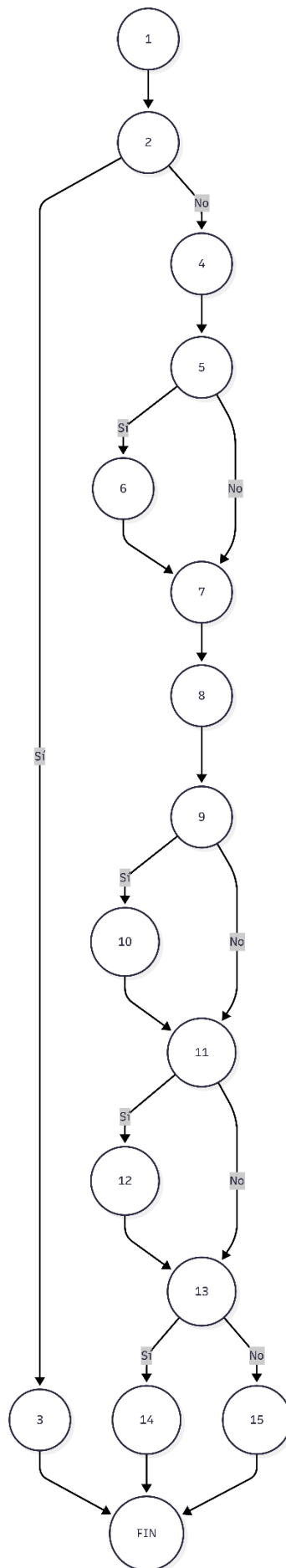
12. DIAGRAMA DE FLUJO (DF)

Diagrama de flujo del requisito 6



13. GRAFO DE FLUJO (GF)

Grafo de flujo del requisito 6



14. IDENTIFICACIÓN DE LAS RUTAS (Camino básico)

Determinar en base al GF del numeral 4

RUTAS

Ruta 1

Camino:

1 → 2 → 3 → 4 → 5 → 6 → 7 → 10

Descripción:

No existen movimientos registrados, por lo que se muestra un mensaje indicando que no hay historial.

Datos de prueba:

- movimientos = []

Ruta 2

Camino:

1 → 2 → 3 → 4 → 5 → 6 → 8 → 9 → 10

Descripción:

Existen movimientos registrados y se muestran en la tabla del historial.

Datos de prueba:

- movimientos =
- [
- {
- "fecha_hora": "01/01/2026 10:30:00",
- "usuario": "Admin",
- "accion": "Ingreso de medicamento"
- }
-]

Ruta 3

Camino:

1 → 2 → 3 → 4 → 5 → 6 → 8 → 9 → 8 → 9 → 10

Descripción:

Existen varios movimientos registrados y el ciclo for se ejecuta más de una vez.

Datos de prueba:

- movimientos con 2 o más registros

15. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

Nodos (N): Son todos los círculos numerados.

Nodos predicados (P): Son los nodos de decisión, que tienen más de una salida: -

A (aristas): Contando todas las flechas entre nodos.

- $V(G) = \text{número de nodos predicados(decisiones)} + 1$
- $(V/G) = P + 1$
- $V(G) = A - N + 2$

DONDE

P: Número de nodos predicado

A: Número de aristas

N: Número de nodos

Nodos (N): 10

(Nodos numerados del 1 al 10)

Nodos predicados (P): 2

Son los nodos de decisión:

- Nodo 6: ¿movements está vacío? (if not movements)
- Nodo 8: Ciclo for mov in movements

Aristas (A): 11

(considerando bifurcación del if y repetición del for)

Cálculo de la Complejidad Ciclomática

Método 1

$$V(G) = P + 1$$

$$V(G) = 2 + 1 = 3$$

Método 2

$$V(G) = A - N + 2$$

$$V(G) = 11 - 10 + 2 = 3$$

RESULTADO FINAL

La complejidad ciclomática del método show_history() es: 3

Esto coincide con el número de rutas independientes identificadas.