1. REQ001--- Acceso al sistema

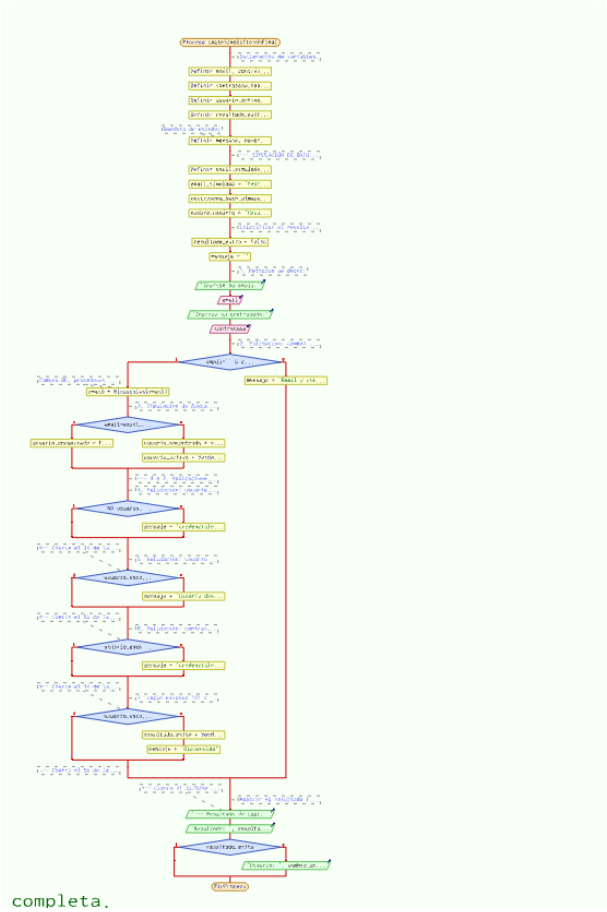
El sistema deberá tener una interfaz gráfica con acceso por credenciales únicas, El usuario podrá crear un usuario y contraseña, además deberá existir un usuario administrador que pueda restringir accesos y funciones del sistema, la validación de estos tendrá un límite de intento para preservar la seguridad del sistema

2. CÓDIGO FUENTE

Lógica de Autenticación

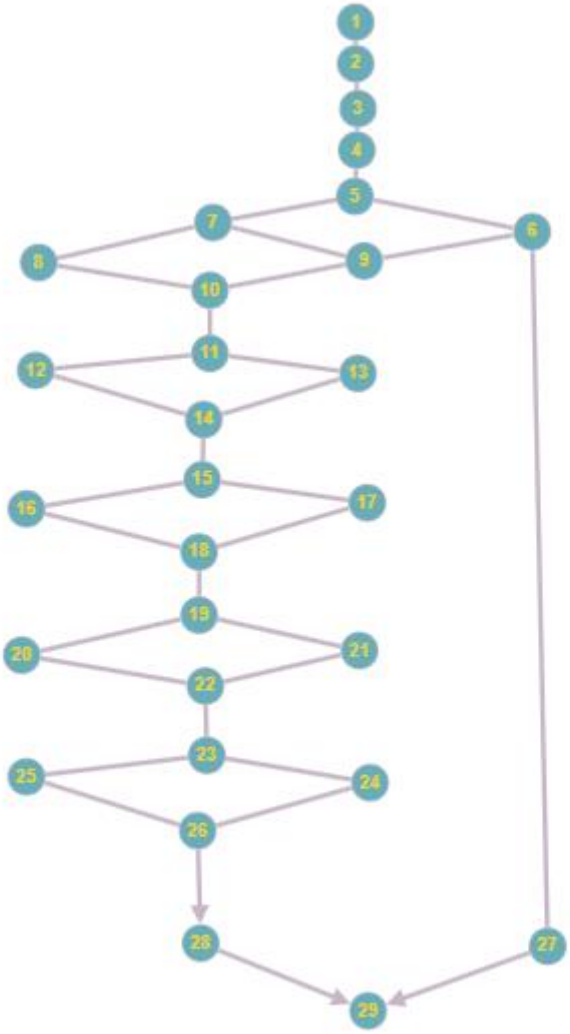
```python
class AuthService:
    _current_user: Optional[User] = None

    @staticmethod
    def _hash_password(password: str) -> str:
        return hashlib.sha256(password.encode()).hexdigest()

    @staticmethod
    def _verify_password(password: str, password_hash: str) -> bool:
        return hashlib.sha256(password.encode()).hexdigest() == password_hash

    def login(self, email: str, password: str) -> Tuple[bool, str, Optional[User]]:
        email = email.strip().lower()

        # Validación: campos vacíos
        if not email or not password:
            return False, "Email y contraseña son requeridos", None

        db = get_db()

        # Buscar usuario
        if db.is_demo_mode:
            user = next((u for u in self._demo_users if u.email.lower() == email), None)
        else:
            with db.session_scope() as session:
                user_db = session.query(UserDB).filter(UserDB.email == email).first()
                if user_db:
                    user = User(
                        id=str(user_db.id),
                        email=user_db.email,
                        password_hash=user_db.password_hash,
                        name=user_db.name,
                        role=UserRole(user_db.role),
                        is_active=user_db.is_active
                    )
                else:
                    user = None

        # Validación: usuario no existe
        if not user:
            return False, "Credenciales incorrectas", None

        # Validación: usuario desactivado
        if not user.is_active:
            return False, "Usuario desactivado", None

        # Validación: contraseña incorrecta
        if not self._verify_password(password, user.password_hash):
            return False, "Credenciales incorrectas", None

        # Login exitoso
        self._current_user = user
        return True, "Bienvenido", user
```

3. DIAGRAMA DE FLUJO (DF)



completa.

4. GRAFO DE FLUJO (GF)



5. IDENTIFIACCIÒN DE LAS RUTAS (Camino básico)

RUTAS

Ruta 1: 1 → 2 → 3 → 4 → 5 → 7 → 10 → 11 → 14 → 15 → 18 → 19 → 22 → 23 → 26 → 28

Ruta 2: 1 → 2 → 3 → 4 → 5 → 7 → 10 → 11 → 14 → 15 → 18 → 19 → 22 → 23 → 27 → 28

Ruta 3: 1 → 2 → 3 → 4 → 5 → 7 → 10 → 11 → 14 → 17 → 18 → 19 → 22 → 23 → 26 → 28

Ruta 4: 1 → 2 → 3 → 4 → 5 → 7 → 10 → 11 → 14 → 17 → 18 → 19 → 22 → 23 → 27 → 28

Ruta 5: 1 → 2 → 3 → 4 → 5 → 7 → 10 → 13 → 14 → 15 → 18 → 19 → 22 → 23 → 26 → 28

Ruta 6: 1 → 2 → 3 → 4 → 5 → 7 → 10 → 13 → 14 → 15 → 18 → 19 → 22 → 23 → 27 → 28

Ruta 7: 1 → 2 → 3 → 4 → 5 → 7 → 10 → 13 → 14 → 17 → 18 → 19 → 22 → 23 → 26 → 28

Ruta 8: 1 → 2 → 3 → 4 → 5 → 7 → 10 → 13 → 14 → 17 → 18 → 19 → 22 → 23 → 27 → 28

6. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

Nodos (N): Son todos los círculos numerados.
N = 29

Nodos predicados (P):
Entonces:
P = 7

- V(G) = número de nodos predicados(decisiones)+1
  V/G) = P + 1
  V(G) = 7 + 1
  V(G) = 8

- V(G) = A – N + 2
  V(G) = 35 – 29 +2
  V(G) = 8

DONDE:
P: Número de nodos predicado
A: Número de aristas
N: Número de nodos

1. REQ002--- Carga de archivos

El sistema deberá reflejar un apartado para la carga y eliminación de archivos de donde se saca la información. Una vez ingresado al sistema, se reflejará una ventana en donde se muestren las opciones de carga de archivos, y un botón para la eliminación de archivos en caso de carga errónea

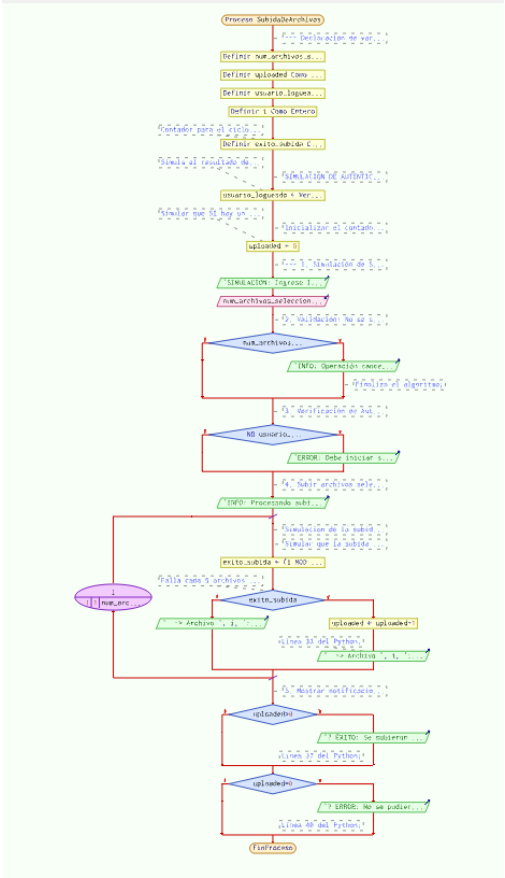
2. CÓDIGO FUENTE

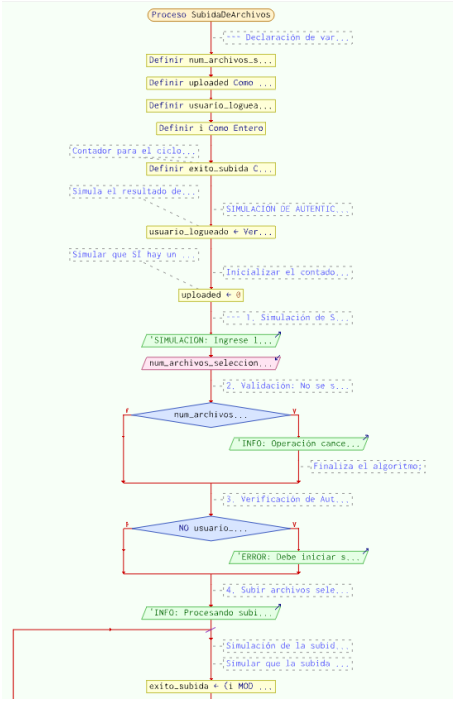
interacción con el usuario

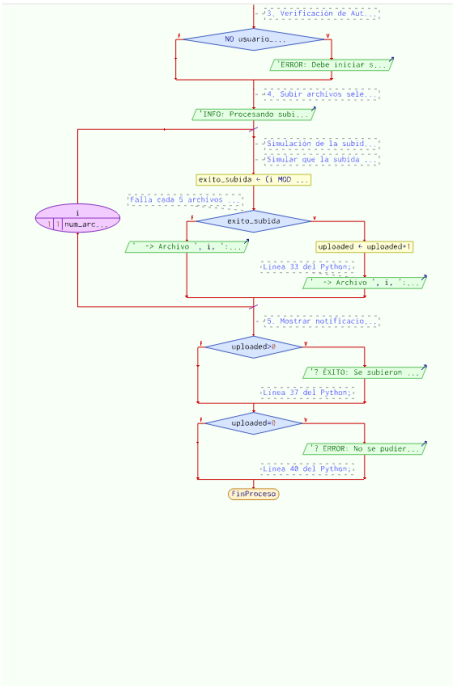
```python
def _on_upload_clicked(self):
    """Maneja clic en subir archivo."""
    # Abrir diálogo para seleccionar archivos PDF
    files, _ = QFileDialog.getOpenFileNames(
        self,
        "Seleccionar archivos PDF",
        "",
        "Archivos PDF (*.pdf)"
    )

    # Validar si no se seleccionaron archivos
    if not files:
        return

    # Verificar si el usuario está autenticado
    auth = get_auth()
    user = auth.current_user

    if not user:
        get_notifications().error("Debe iniciar sesión para subir archivos")
        return

    # Subir archivos seleccionados
    uploaded = 0
    for filepath in files:
        success, message, invoice = self.invoice_service.upload_file(
            filepath,
            user.id,
            user.name,
            user.role.value if hasattr(user.role, 'value') else str(user.role)
        )
        if success:
            uploaded += 1

    # Mostrar notificaciones según el resultado
    if uploaded > 0:
        get_notifications().success(f"Se subieron {uploaded} archivo(s)")
        self._load_files()
    else:
        get_notifications().error("No se pudieron subir los archivos")
```

lógica de negocio para validar, copiar y registrar los archivos subidos.

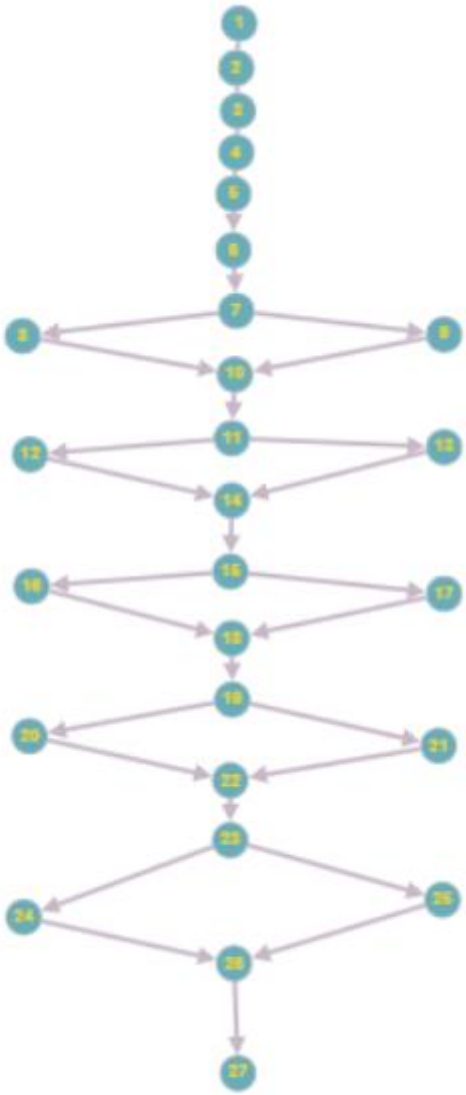
3. DIAGRAMA DE FLUJO (DF)







4. **GRAFO DE FLUJO (GF)**



5. **IDENTIFIACCIÒN DE LAS RUTAS (Camino básico)**

Determinar en base al GF del numeral 4

Ruta 1: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 10 → 11 → 14 → 15 → 18 → 19 → 22 → 23 → 26 → 27

Ruta 2: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 10 → 11 → 12 → 11 → 14 → 15 → 18 → 19 → 22 → 23 → 26 → 27

Ruta 3: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 10 → 11 → 14 → 15 → 18 → 19 → 22 → 23 → 24 → 23 → 26 → 27

Ruta 4: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 10 → 11 → 12 → 11 → 14 → 15 → 18 → 19 → 22 → 23 → 24 → 23 → 26 → 27

Ruta 5: 1 → 2 → 3 → 4 → 5 → 6 → 8 → 10 → 11 → 14 → 15 → 18 → 19 → 22 → 23 → 26 → 27

Ruta 6: 1 → 2 → 3 → 4 → 5 → 6 → 9 → 10 → 11 → 14 → 15 → 18 → 19 → 22 → 23 → 26 → 27

6. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

Nodos (N):
N = 27

Nodos predicados (P):
P = 5

- V(G) = número de nodos predicados(decisiones)+1
  V/G) = P + 1
  V(G) = 5 + 1
  V(G) = 6

- V(G) = A – N + 2
  V(G) = 31 -27 +2
  V(G) = 6

DONDE:
P: Número de nodos predicado
A: Número de aristas
N: Número de nodos

1. **REQ003--- Consuta/Busqueda**

El sistema deberá contener un apartado (interfaz) en donde permita realizar la búsqueda mediante filtros de relevancia. Luego de terminar la búsqueda, el usuario accederá a la interfaz de filtrado, en donde podrá buscar mediante filtros el número de lote, proveedor o producto que se requiera.

2. **CÓDIGO FUENTE**

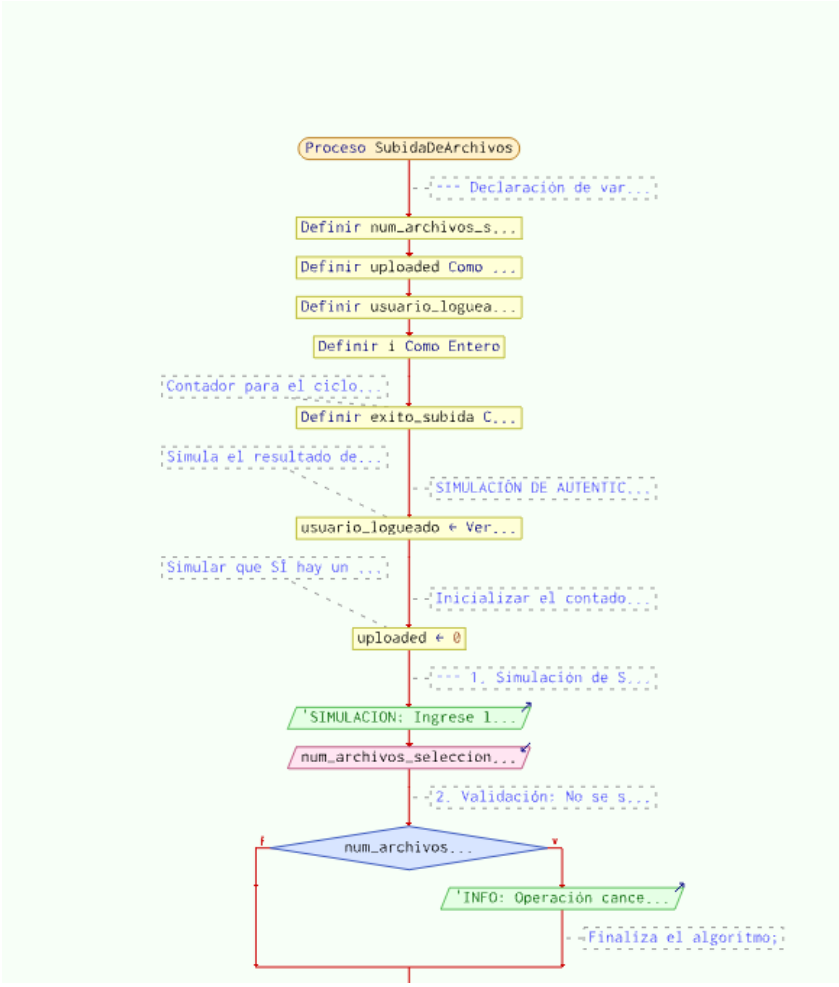
lógica para buscar facturas analizadas con filtros opcionales.

cargar y mostrar los datos de las facturas analizadas en una tabla. También permite aplicar filtros.

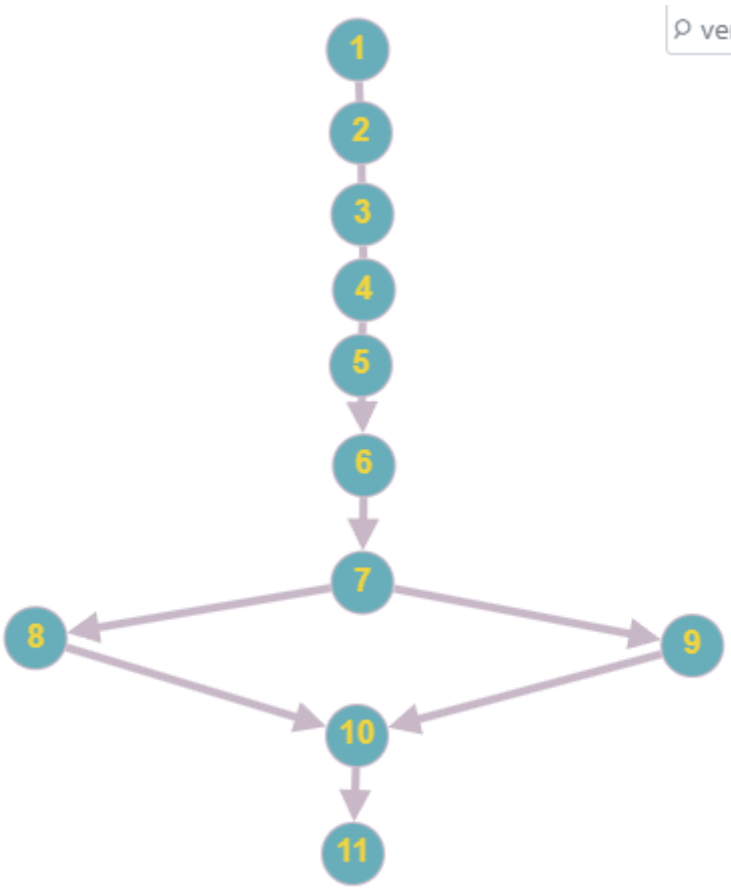
```python
def _load_data(self, filter_batch: str = "", filter_company: str = "", filter_product: str = ""):
    """Carga los datos en la tabla."""
    # Obtener datos filtrados
    if filter_batch or filter_company or filter_product:
        invoices = self.invoice_service.search_invoices(
            batch_number=filter_batch,
            company=filter_company,
            product_name=filter_product
        )
    else:
        invoices = self.invoice_service.get_completed_invoices()

    self._current_data = invoices

    # Actualizar tabla
    self.table.setRowCount(len(invoices))

    for row, invoice in enumerate(invoices):
        # Archivo
        self.table.setItem(row, 0, QTableWidgetItem(invoice.filename))

        # Subido por
        self.table.setItem(row, 1, QTableWidgetItem(invoice.uploaded_by_name))

        # Rol
        role_text = self._format_role(invoice.uploaded_by_role)
        role_item = QTableWidgetItem(role_text)
        role_item.setTextAlignment(Qt.AlignmentFlag.AlignCenter)
        self.table.setItem(row, 2, role_item)
```

campo para aplicar filtros y una tabla para mostrar los resultados

3. **DIAGRAMA DE FLUJO (DF)**



4. GRAFO DE FLUJO (GF)



5. IDENTIFIACCIÒN DE LAS RUTAS (Camino básico)

Determinar en base al GF del numeral 4

RUTAS

Ruta 1: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 10 → 11

Ruta 2: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 9 → 10 → 11

6. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

Nodos (N):.
N = 11

Nodos predicados (P):
P = 1

- V(G) = número de nodos predicados(decisiones)+1
  V/G) = P + 1

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

- V(G) = A – N + 2
  V(G) = 11 -11 +2
  V(G) =2

DONDE:
P: Número de nodos predicado
A: Número de aristas
N: Número de nodos

1. REQ004--- Generación Informes

El sistema deberá contener un apartado (interfaz) en donde permita realizar la búsqueda mediante filtros de relevancia. Luego de terminar la búsqueda, el usuario accederá a la interfaz de filtrado, en donde podrá buscar mediante filtros el número de lote, proveedor o producto que se requiera.

2. CÓDIGO FUENTE

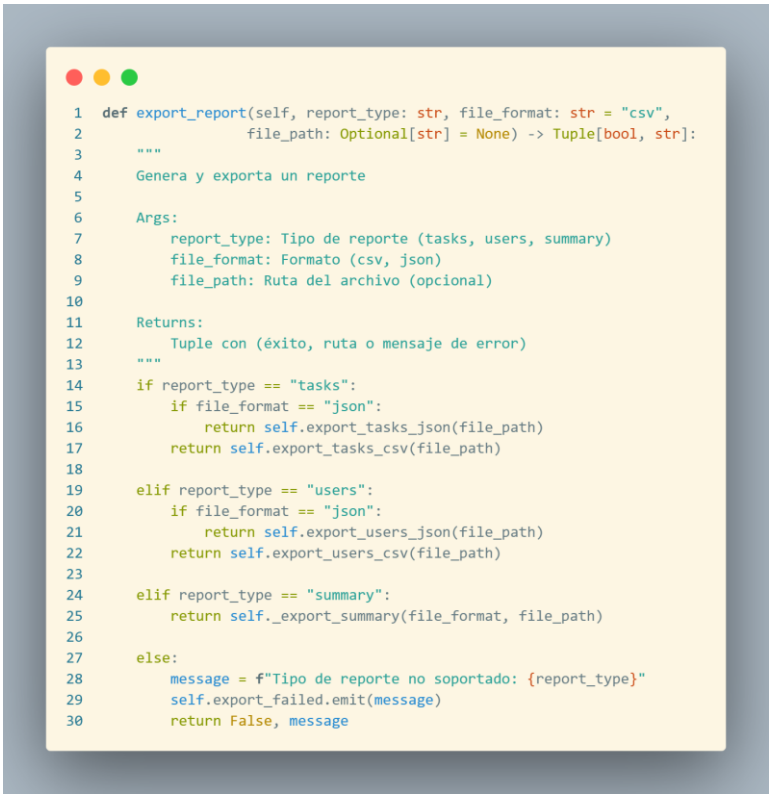
lógica para exportar datos a diferentes formatos, como PDF

```python
@staticmethod
def export_to_pdf(
    data: List[Dict[str, Any]],
    filepath: str,
    title: str = "Reporte"
) -> Tuple[bool, str]:
    """
    Exporta datos a archivo PDF.

    Args:
        data: Lista de diccionarios con datos
        filepath: Ruta del archivo destino
        title: Título del reporte

    Returns:
        Tuple (éxito, mensaje)
    """
    if not data:
        return False, "No hay datos para exportar"

    try:
        from reportlab.lib import colors
        from reportlab.lib.pagesizes import letter, landscape
        from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Paragraph, Spacer
        from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle

        doc = SimpleDocTemplate(
            filepath,
            pagesize=landscape(letter),
            rightMargin=30,
            leftMargin=30,
            topMargin=30,
            bottomMargin=30
        )

        elements = []
        styles = getSampleStyleSheet()

        # Título
        title_style = ParagraphStyle(
            'CustomTitle',
            parent=styles['Heading1'],
            fontSize=18,
            spaceAfter=20,
            textColor=colors.HexColor('#E94560')
        )
        elements.append(Paragraph(title, title_style))

        # Tabla
        headers = list(data[0].keys())
        table_data = [headers]
        for row in data:
            table_data.append([str(row.get(h, "")) for h in headers])

        table = Table(table_data, repeatRows=1)
        table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#E94560')),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
            ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
            ('GRID', (0, 0), (-1, -1), 1, colors.lightgrey)
        ]))
        elements.append(table)

        # Generar PDF
        doc.build(elements)
        return True, f"Exportados {len(data)} registros a PDF"
    except Exception as e:
        return False, f"Error al exportar: {str(e)}"
```

lógica para exportar datos a diferentes formatos, como Excel

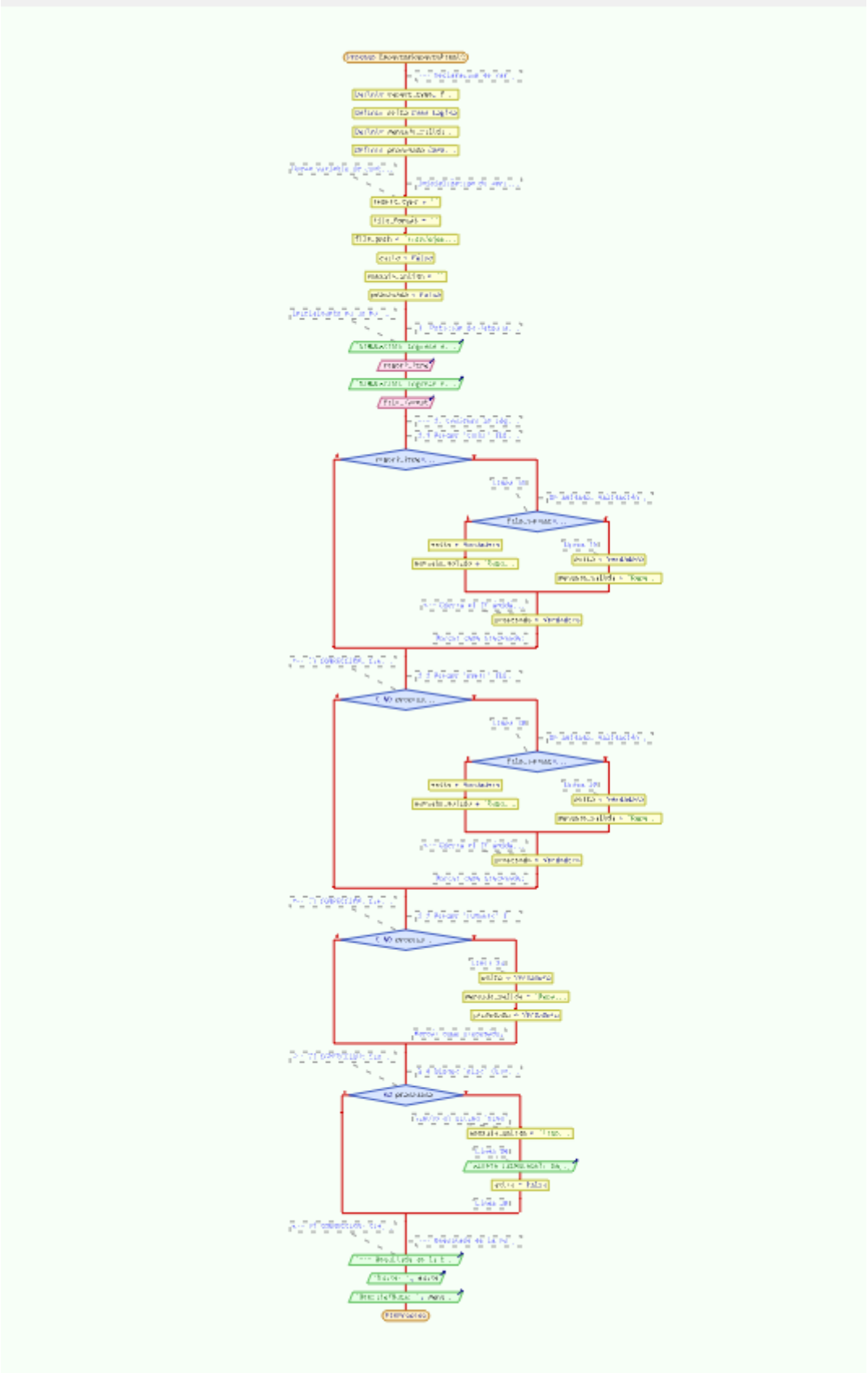
```python
@staticmethod
def export_to_excel(
    data: List[Dict[str, Any]],
    filepath: str
) -> Tuple[bool, str]:
    """
    Exporta datos a archivo Excel.

    Args:
        data: Lista de diccionarios con datos
        filepath: Ruta del archivo destino

    Returns:
        Tuple (éxito, mensaje)
    """
    if not data:
        return False, "No hay datos para exportar"

    try:
        from openpyxl import Workbook
        wb = Workbook()
        ws = wb.active
        ws.title = "Reporte"

        # Escribir encabezados
        headers = list(data[0].keys())
        ws.append(headers)

        # Escribir datos
        for row in data:
            ws.append([row.get(h, "") for h in headers])

        wb.save(filepath)
        return True, f"Exportados {len(data)} registros a Excel"
    except Exception as e:
        return False, f"Error al exportar: {str(e)}"
```

controlador maneja la lógica de exportación y conecta la vista con el servicio.

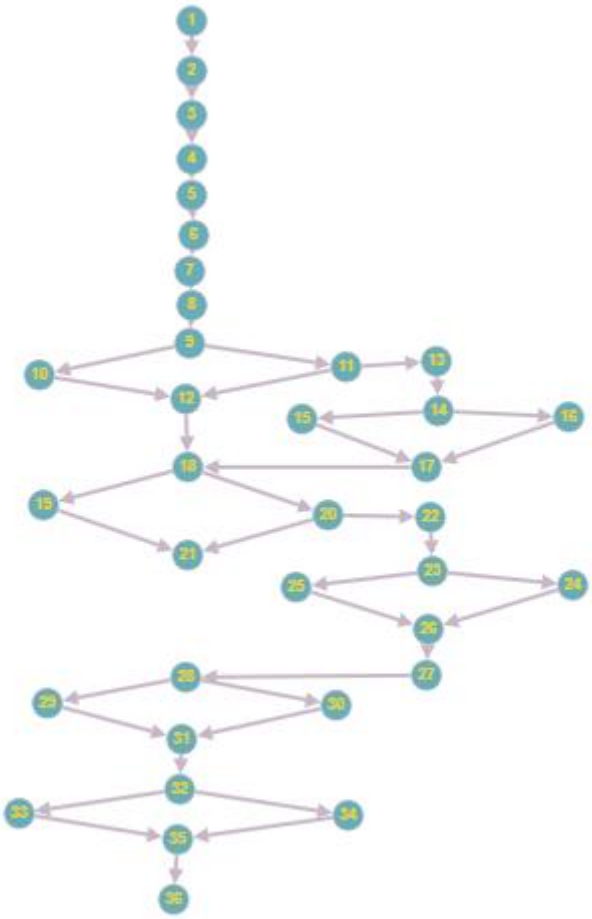


```python
def export_report(self, report_type: str, file_format: str = "csv",
                  file_path: Optional[str] = None) -> Tuple[bool, str]:
    """
    Genera y exporta un reporte

    Args:
        report_type: Tipo de reporte (tasks, users, summary)
        file_format: Formato (csv, json)
        file_path: Ruta del archivo (opcional)

    Returns:
        Tuple con (éxito, ruta o mensaje de error)
    """
    if report_type == "tasks":
        if file_format == "json":
            return self.export_tasks_json(file_path)
        return self.export_tasks_csv(file_path)

    elif report_type == "users":
        if file_format == "json":
            return self.export_users_json(file_path)
        return self.export_users_csv(file_path)

    elif report_type == "summary":
        return self._export_summary(file_format, file_path)

    else:
        message = f"Tipo de reporte no soportado: {report_type}"
        self.export_failed.emit(message)
        return False, message
```

3. DIAGRAMA DE FLUJO (DF)



4. GRAFO DE FLUJO (GF)



5. IDENTIFIACCIÒN DE LAS RUTAS (Camino básico)

RUTAS

Ruta 1: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 12 → 13 → 16 → 18 → 19 → 22 → 25 → 27 → 28 → 31 → 32 → 35 → 36

Ruta 2: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11 → 12 → 13 → 16 → 18 → 19 → 22 → 25 → 27 → 28 → 31 → 32 → 35 → 36

Ruta 3: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 12 → 14 → 13 → 16 → 18 → 19 → 22 → 25 → 27 → 28 → 31 → 32 → 35 → 36

Ruta 4: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 12 → 13 → 15 → 16 → 18 → 19 → 22 → 25 → 27 → 28 → 31 → 32 → 35 → 36

Ruta 5: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 12 → 13 → 16 → 17 → 16 → 18 → 19 → 22 → 25 → 27 → 28 → 31 → 32 → 35 → 36

Ruta 6: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 12 → 13 → 16 → 18 → 20 → 19 → 22 → 25 → 27 → 28 → 31 → 32 → 35 → 36

Ruta 7: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 12 → 13 → 16 → 18 → 19 → 21 → 22 → 25 → 27 → 28 → 31 → 32 → 35 → 36

6. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

Nodos (N): Son todos los círculos numerados.
N = 36

Nodos predicados (P): Son los nodos de decisión, que tienen más de una salida:
P = 6

- V(G) = número de nodos predicados(decisiones)+1
  V/G) = P + 1
  V(G) = 6 + 1
  V(G) = 7

- V(G) = A – N + 2
  V(G) = 41 -36 +2
  V(G) = 7

DONDE:
P: Número de nodos predicado
A: Número de aristas
N: Número de nodos