

## 1. REQ001--- Acceso al sistema

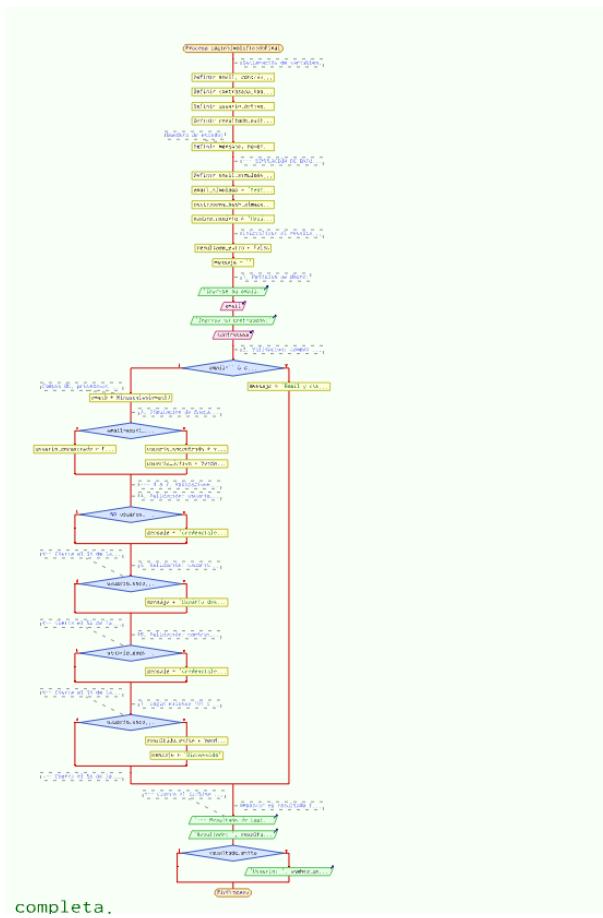
El sistema deberá tener una interfaz gráfica con acceso por credenciales únicas, El usuario podrá crear un usuario y contraseña, además deberá existir un usuario administrador que pueda restringir accesos y funciones del sistema, la validación de estos tendrá un límite de intento para preservar la seguridad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

## 2. CÓDIGO FUENTE

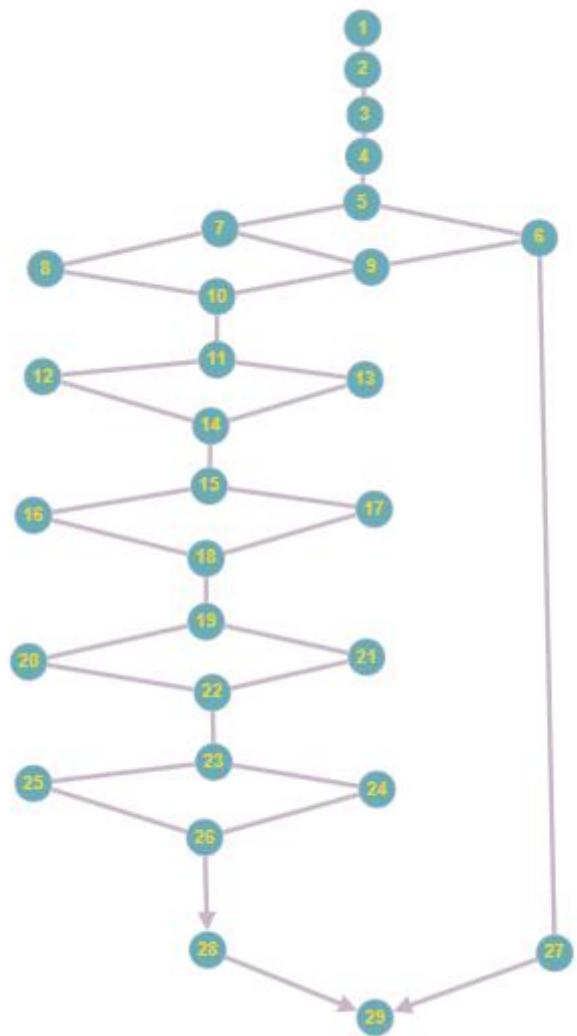
### Lógica de Autenticación

```
● ○ ■
1  class AuthService:
2      _current_user: Optional[User] = None
3
4      @staticmethod
5      def _hash_password(password: str) -> str:
6          return hashlib.sha256(password.encode()).hexdigest()
7
8      @staticmethod
9      def _verify_password(password: str, password_hash: str) -> bool:
10         return hashlib.sha256(password.encode()).hexdigest() == password_hash
11
12     def login(self, email: str, password: str) -> Tuple[bool, str, Optional[User]]:
13         email = email.strip().lower()
14
15         # Validación: campos vacíos
16         if not email or not password:
17             return False, "Email y contraseña son requeridos", None
18
19         db = get_db()
20
21         # Buscar usuario
22         if db.is_demo_mode:
23             user = next((u for u in self._demo_users if u.email.lower() == email), None)
24         else:
25             with db.session_scope() as session:
26                 user_db = session.query(UserDB).filter(UserDB.email == email).first()
27                 if user_db:
28                     user = User(
29                         id=str(user_db.id),
30                         email=user_db.email,
31                         password_hash=user_db.password_hash,
32                         name=user_db.name,
33                         role=UserRole(user_db.role),
34                         is_active=user_db.is_active
35                     )
36                 else:
37                     user = None
38
39         # Validación: usuario no existe
40         if not user:
41             return False, "Credenciales incorrectas", None
42
43         # Validación: usuario desactivado
44         if not user.is_active:
45             return False, "Usuario desactivado", None
46
47         # Validación: contraseña incorrecta
48         if not self._verify_password(password, user.password_hash):
49             return False, "Credenciales incorrectas", None
50
51         # Login exitoso
52         self._current_user = user
53         return True, "Bienvenido", user
```

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



## 4. GRAFO DE FLUJO (GF)



## 5. IDENTIFICACIÓN DE LAS RUTAS (Camino básico)

### RUTAS

**R1:** 1-2-3-4-5-14 *Campos vacíos*

**R2:** 1-2-3-4-6-7-14 *Usuario no encontrado*

**R3:** 1-2-3-4-6-8-9-14 *Usuario desactivado*

**R4:** 1-2-3-4-6-8-10-11-14 *Password incorrecto*

**R5:** 1-2-3-4-6-8-10-12-13-14 *Login exitoso*

## 6. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

**Nodos (N):** Son todos los círculos numerados.

$N = 14$

**Nodos predicados (P):**

Entonces:

$P = 4$

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

$$V/G = P + 1$$

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

$$V(G) = 5$$

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

$$V(G) = 17 - 14 + 2$$

$$V(G) = 5$$

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

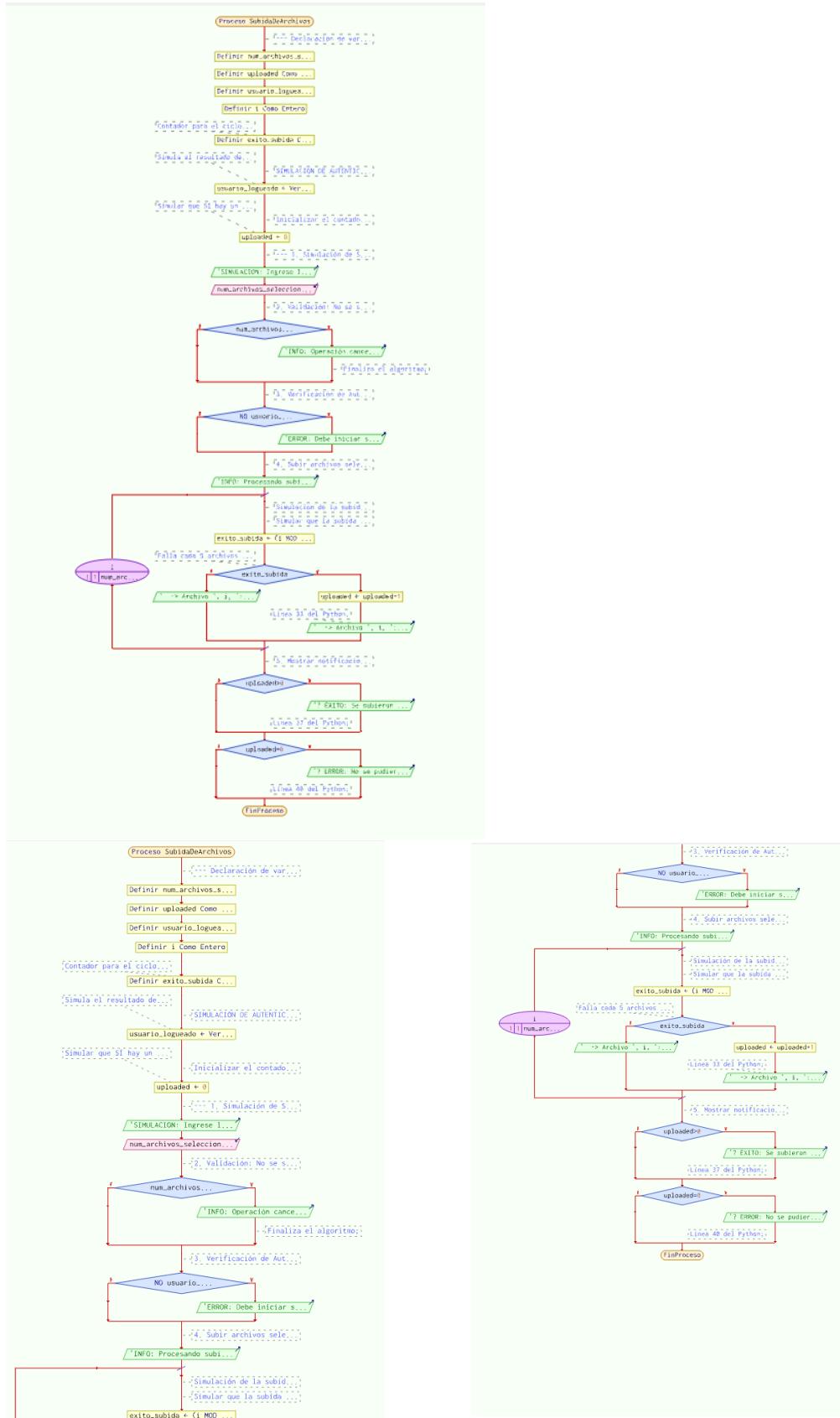
```

1  def _on_upload_clicked(self):
2      """Maneja clic en subir archivo."""
3      # Abrir diálogo para seleccionar archivos PDF
4      files, _ = QFileDialog.getOpenFileNames(
5          self,
6          "Seleccionar archivos PDF",
7          "",
8          "Archivos PDF (*.pdf)"
9      )
10
11     # Validar si no se seleccionaron archivos
12     if not files:
13         return
14
15     # Verificar si el usuario está autenticado
16     auth = get_auth()
17     user = auth.current_user
18
19     if not user:
20         get_notifications().error("Debe iniciar sesión para subir archivos")
21         return
22
23     # Subir archivos seleccionados
24     uploaded = 0
25     for filepath in files:
26         success, message, invoice = self.invoice_service.upload_file(
27             filepath,
28             user.id,
29             user.name,
30             user.role.value if hasattr(user.role, 'value') else str(user.role)
31         )
32         if success:
33             uploaded += 1
34
35     # Mostrar notificaciones según el resultado
36     if uploaded > 0:
37         get_notifications().success(f"Se subieron {uploaded} archivo(s)")
38         self._load_files()
39     else:
40         get_notifications().error("No se pudieron subir los archivos")

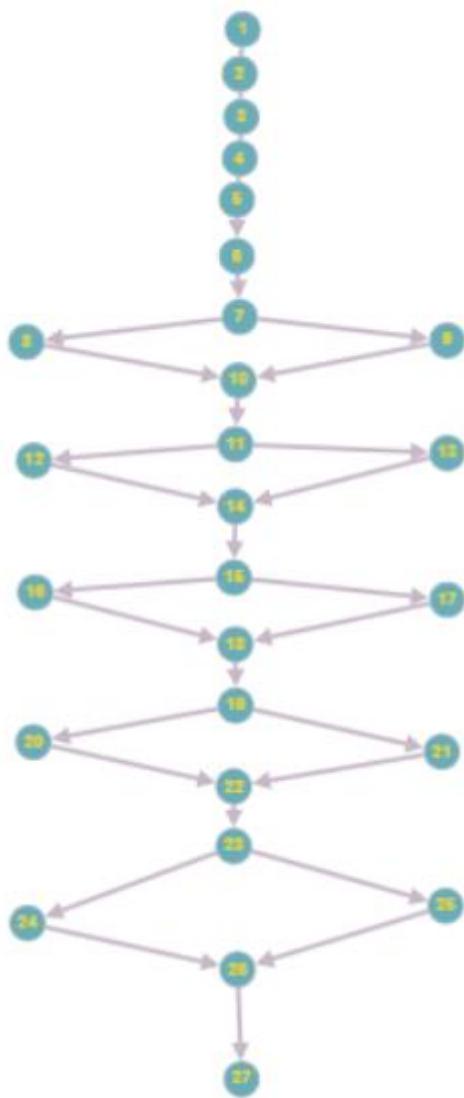
```

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

### 3. DIAGRAMA DE FLUJO (DF)



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



## 5. IDENTIFICACIÓN DE LAS RUTAS (Camino básico)

Determinar en base al GF del numeral 4

### RUTAS

**R1:** 1-2-3-4-5-20 *No se hizo clic en subir*

**R2:** 1-2-3-4-6-7-8-20 *No se seleccionaron archivos*

**R3:** 1-2-3-4-6-7-9-10-8-20 *Usuario no autenticado*

**R4:** 1-2-3-4-6-7-9-10-11-12-13-15-19-17-20 *Archivo sin extensión válida o no PDF - Error subida*

**R5:** 1-2-3-4-6-7-9-10-11-12-13-15-18-17-20 *Archivo sin extensión válida pero subida exitosa*

**R6:** 1-2-3-4-6-7-9-10-11-12-14-15-19-17-20 *Archivo no es PDF - Error*

**R7:** 1-2-3-4-6-7-9-10-11-12-14-15-16-19-17-20 *Archivo es PDF pero falla la subida*

**R8:** 1-2-3-4-6-7-9-10-11-12-14-15-16-18-17-20 *Subida exitosa de PDF*

## 6. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

**Nodos (N):** Son todos los círculos numerados.

$$N = 20$$

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

$$P = 4$$

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

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

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

$$V(G) = 5$$

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

$$V(G) = 23 - 20 + 2$$

$$V(G) = 5$$

DONDE:

**P:** Número de nodos predicho

**A:** Número de aristas

**N:** Número de nodos

## **1. REQ003--- Consulta/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**

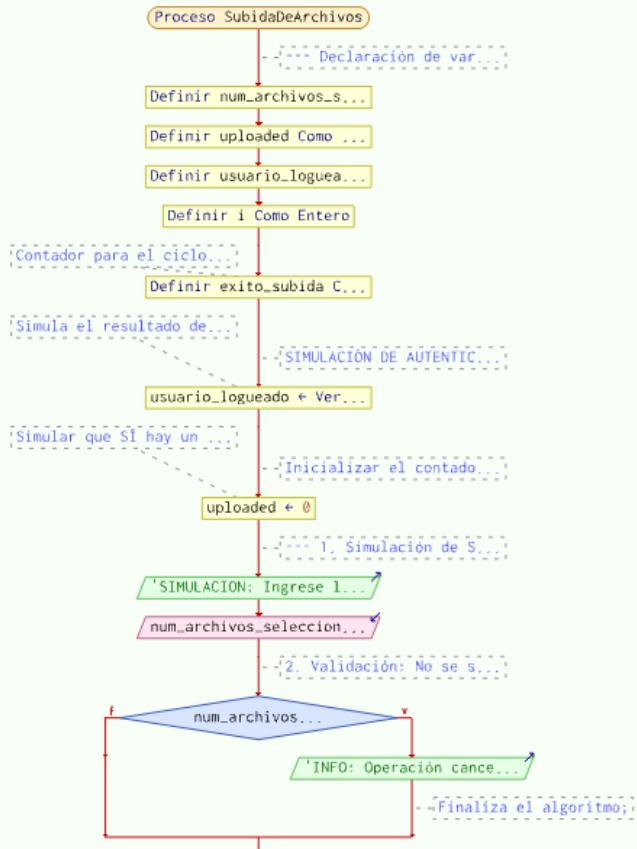
**lógica para buscar facturas analizadas con filtros opcionales.**

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

```
 1  def _load_data(self, filter_batch: str = "", filter_company: str = "", filter_product: str = ""):  
 2      """Carga los datos en la tabla."""  
 3      # Obtener datos filtrados  
 4      if filter_batch or filter_company or filter_product:  
 5          invoices = self.invoice_service.search_invoices(  
 6              batch_number=filter_batch,  
 7              company=filter_company,  
 8              product_name=filter_product  
 9          )  
10     else:  
11         invoices = self.invoice_service.get_completed_invoices()  
12  
13     self._current_data = invoices  
14  
15     # Actualizar tabla  
16     self.table.setRowCount(len(invoices))  
17  
18     for row, invoice in enumerate(invoices):  
19         # Archivo  
20         self.table.setItem(row, 0, QTableWidgetItem(invoice.filename))  
21  
22         # Subido por  
23         self.table.setItem(row, 1, QTableWidgetItem(invoice.uploaded_by_name))  
24  
25         # Rol  
26         role_text = self._format_role(invoice.uploaded_by_role)  
27         role_item = QTableWidgetItem(role_text)  
28         role_item.setTextAlignment(Qt.AlignmentFlag.AlignCenter)  
29         self.table.setItem(row, 2, role_item)
```

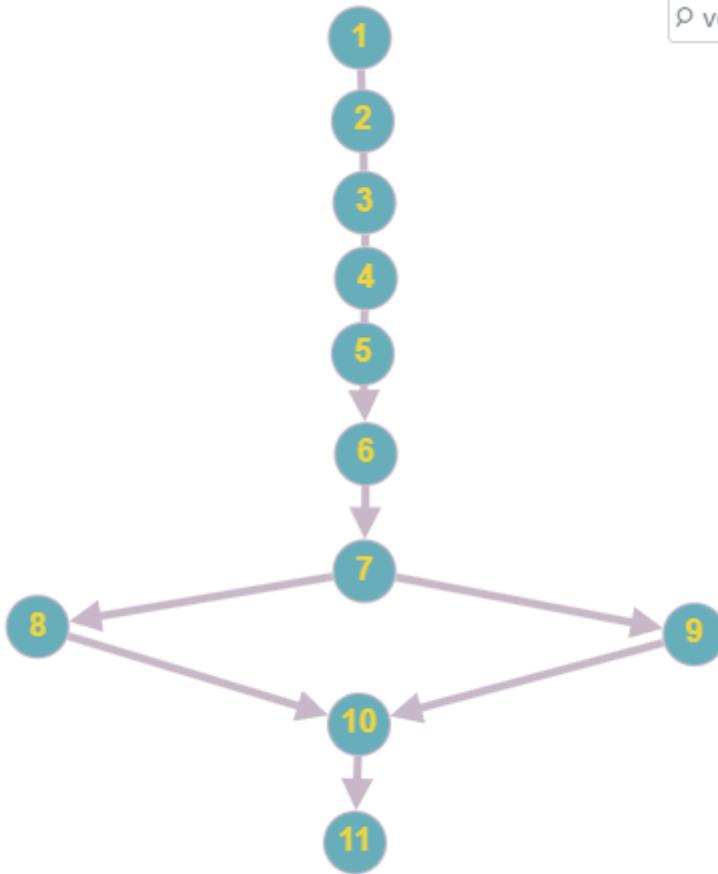
**campo para aplicar filtros y una tabla para mostrar los resultados**

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



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

vertex n°



## 5. IDENTIFICACIÓN DE LAS RUTAS (Camino básico)

Determinar en base al GF del numeral 4

### RUTAS

**R1:** 1-2-3-4-5-6-7-8-10-11-13-14-15-16-17-16-19-20-21

**R2:** 1-2-3-4-5-6-7-8-10-11-13-14-18-19-20-21

**R3:** 1-2-3-4-5-6-7-9-10-12-13-14-15-16-17-16-19-20-21

**R4:** 1-2-3-4-5-6-7-9-10-12-13-14-18-19-20-21

## 6. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

**Nodos (N):** Son todos los círculos numerados.

$N = 21$

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

$P = 4$

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

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

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

$$V(G) = 5$$

➤  $V(G) = A - N + 2$

$$V(G) = 24 - 21 + 2$$

$$V(G) = 5$$

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**

```

1  @staticmethod
2  def export_to_pdf(
3      data: List[Dict[str, Any]],
4      filepath: str,
5      title: str = "Reporte"
6  ) -> Tuple[bool, str]:
7      """
8          Exporta datos a archivo PDF.
9
10     Args:
11         data: Lista de diccionarios con datos
12         filepath: Ruta del archivo destino
13         title: Titulo del reporte
14
15     Returns:
16         Tuple (éxito, mensaje)
17     """
18     if not data:
19         return False, "No hay datos para exportar"
20
21     try:
22         from reportlab.lib import colors
23         from reportlab.lib.pagesizes import letter, landscape
24         from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Paragraph, Spacer
25         from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
26
27         doc = SimpleDocTemplate(
28             filepath,
29             pagesize=landscape(letter),
30             rightMargin=30,
31             leftMargin=30,
32             topMargin=30,
33             bottomMargin=30
34         )
35
36         elements = []
37         styles = getSampleStyleSheet()
38
39         # Título
40         title_style = ParagraphStyle(
41             'CustomTitle',
42             parent=styles['Heading1'],
43             fontSize=18,
44             spaceAfter=20,
45             textColor=colors.HexColor('#E94560')
46         )
47         elements.append(Paragraph(title, title_style))
48
49         # Tabla
50         headers = list(data[0].keys())
51         table_data = [headers]
52         for row in data:
53             table_data.append([str(row.get(h, "")) for h in headers])
54
55         table = Table(table_data, repeatRows=1)
56         table.setStyle(TableStyle([
57             ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#E94560')),
58             ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
59             ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
60             ('GRID', (0, 0), (-1, -1), 1, colors.lightgrey)
61         ]))
62         elements.append(table)
63
64         # Generar PDF
65         doc.build(elements)
66         return True, f"Exportados {len(data)} registros a PDF"
67     except Exception as e:
68         return False, f"Error al exportar: {str(e)}"

```

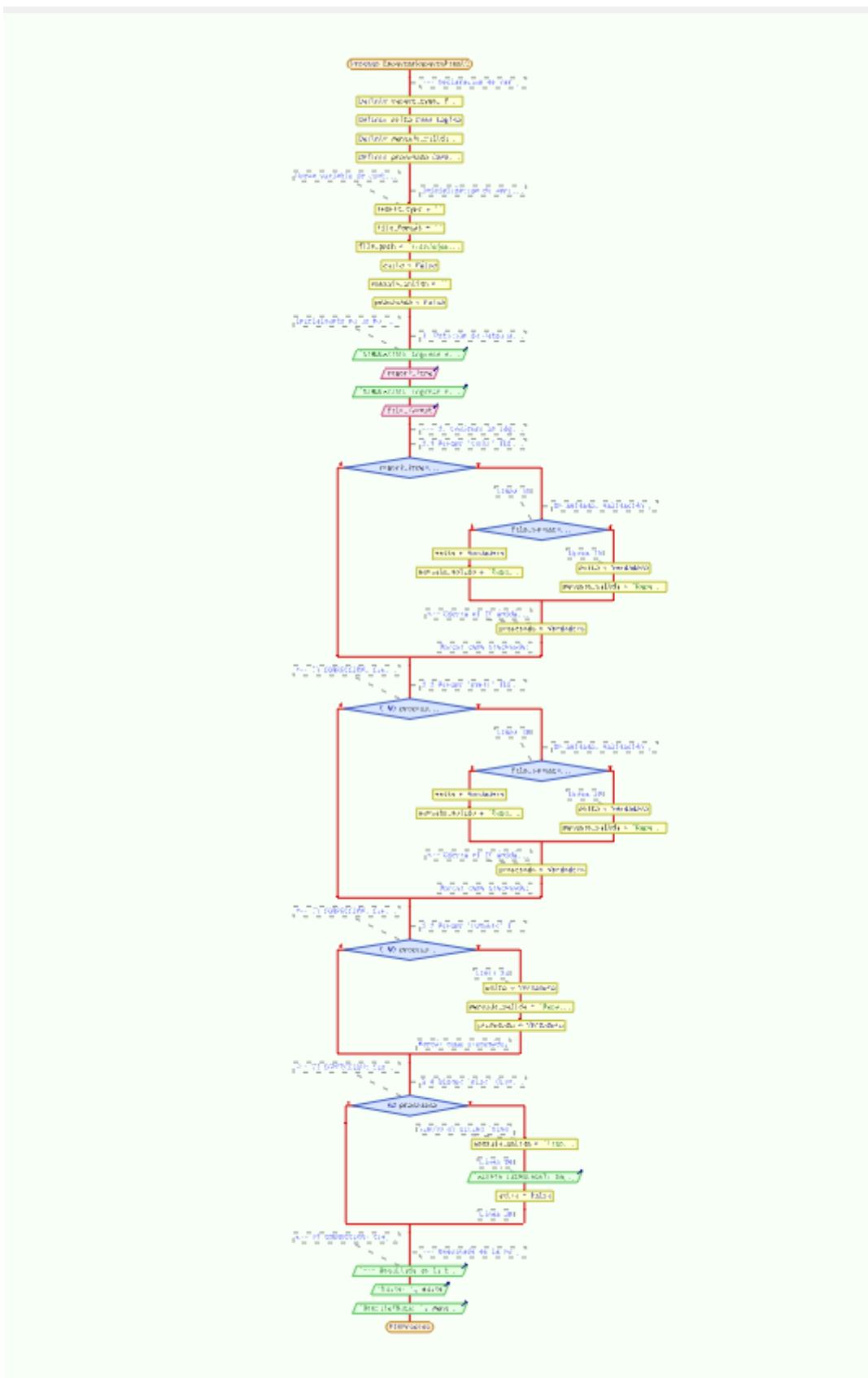
**lógica para exportar datos a diferentes formatos, como Excel**

```
 1  @staticmethod
 2  def export_to_excel(
 3      data: List[Dict[str, Any]],
 4      filepath: str
 5  ) -> Tuple[bool, str]:
 6      """
 7          Exporta datos a archivo Excel.
 8
 9      Args:
10          data: Lista de diccionarios con datos
11          filepath: Ruta del archivo destino
12
13      Returns:
14          Tuple (éxito, mensaje)
15      """
16
17      if not data:
18          return False, "No hay datos para exportar"
19
20      try:
21          from openpyxl import Workbook
22          wb = Workbook()
23          ws = wb.active
24          ws.title = "Reporte"
25
26          # Escribir encabezados
27          headers = list(data[0].keys())
28          ws.append(headers)
29
30          # Escribir datos
31          for row in data:
32              ws.append([row.get(h, "") for h in headers])
33
34          wb.save(filepath)
35          return True, f"Exportados {len(data)} registros a Excel"
36      except Exception as e:
37          return False, f"Error al exportar: {str(e)}"
```

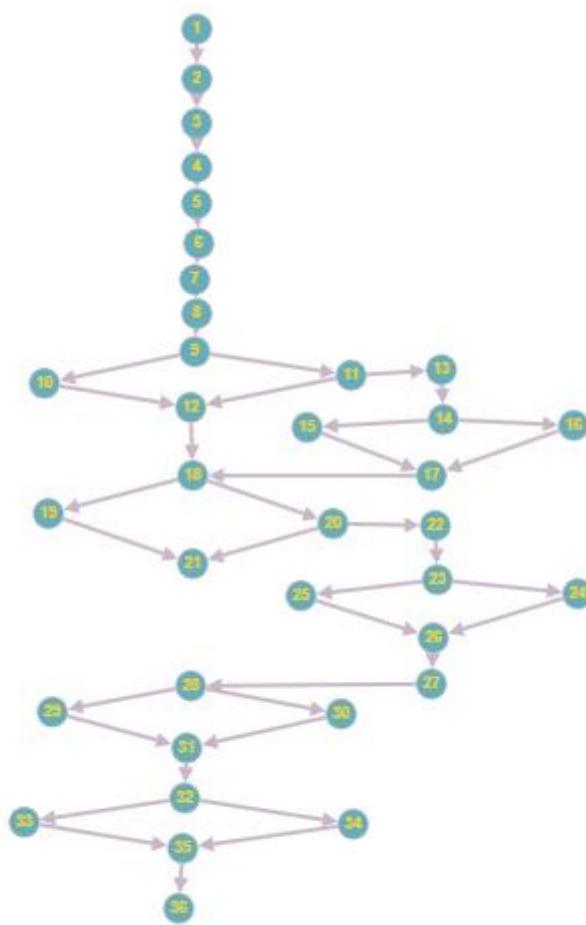
controlador maneja la lógica de exportación y conecta la vista con el servicio.

```
 1  def export_report(self, report_type: str, file_format: str = "csv",
 2                      file_path: Optional[str] = None) -> Tuple[bool, str]:
 3      """
 4          Genera y exporta un reporte
 5
 6      Args:
 7          report_type: Tipo de reporte (tasks, users, summary)
 8          file_format: Formato (csv, json)
 9          file_path: Ruta del archivo (opcional)
10
11      Returns:
12          Tuple con (éxito, ruta o mensaje de error)
13      """
14
15      if report_type == "tasks":
16          if file_format == "json":
17              return self.export_tasks_json(file_path)
18          return self.export_tasks_csv(file_path)
19
20      elif report_type == "users":
21          if file_format == "json":
22              return self.export_users_json(file_path)
23          return self.export_users_csv(file_path)
24
25      elif report_type == "summary":
26          return self._export_summary(file_format, file_path)
27
28      else:
29          message = f"Tipo de reporte no soportado: {report_type}"
30          self.export_failed.emit(message)
31
32      return False, message
```

### 3. DIAGRAMA DE FLUJO (DF)



### 4. GRAFO DE FLUJO (GF)



## 5. IDENTIFICACIÓN DE LAS RUTAS (Camino básico)

### RUTAS

**R1:** 1-2-3-4-5-6-7-8-18-19-20-22 (Selecciona PDF, genera exitosamente, muestra mensaje de éxito)

**R2:** 1-2-3-4-5-9-10-11-18-19-20-22 (Selecciona Excel, genera exitosamente, muestra mensaje de éxito)

**R3:** 1-2-3-4-5-12-13-14-18-19-20-22 (Selecciona CSV, genera exitosamente, muestra mensaje de éxito)

**R4:** 1-2-3-4-5-15-16-18-19-20-22 (Selecciona JSON, genera exitosamente, muestra mensaje de éxito)

**R5:** 1-2-3-4-5-17-18-19-21-22 (Selecciona opción inválida, no genera informe, muestra mensaje de error)

## 6. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

**Nodos (N):** Son todos los círculos numerados.

$$N = 22$$

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

$$P = 5$$

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

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

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

$$V(G) = 3$$

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

$$V(G) = 26 - 22 + 2$$

$$V(G) = 6$$

DONDE:

**P:** Número de nodos predicho

**A:** Número de aristas

**N:** Número de nodos