

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

Extracción de texto

```
1 def extract_text_from_pdf(self, pdf_path: str) -> str:
2     """
3     Extrae texto de un PDF usando PyPDF2 o pdfplumber.
4     """
5     try:
6         # Intentar con PyPDF2 primero
7         try:
8             import PyPDF2
9             with open(pdf_path, 'rb') as file:
10                 reader = PyPDF2.PdfReader(file)
11                 text = ""
12                 for page in reader.pages:
13                     text += page.extract_text() + "\n"
14                 if text.strip():
15                     return text
16         except Exception as e:
17             logger.warning(f"PyPDF2 falló: {e}")
18
19         # Fallback a pdfplumber
20         try:
21             import pdfplumber
22             text = ""
23             with pdfplumber.open(pdf_path) as pdf:
24                 for page in pdf.pages:
25                     page_text = page.extract_text()
26                     if page_text:
27                         text += page_text + "\n"
28
29                 # Para Merck, también extraer tablas
30                 if self.provider_id == 'merck':
31                     tables = page.extract_tables()
32                     if tables:
33                         for table in tables:
34                             for row in table:
35                                 if row:
36                                     text += " | ".join([str(cell) if cell else "" for cell in row]) + "\n"
37
38             return text
39         except Exception as e:
40             logger.warning(f"pdfplumber falló: {e}")
41
42     except Exception as e:
43         logger.error(f"Error extrayendo texto del PDF: {e}")
44         return ""
45     return ""
```

Identificación del proveedor

```
1 def identify_provider(self, text: str) -> str:
2     """
3     Identifica el proveedor buscando firmas en el texto.
4     """
5     text_lower = text.lower()
6
7     for provider_id, config in PROVIDER_SIGNATURES.items():
8         if provider_id == 'default':
9             continue
10
11         for keyword in config['keywords']:
12             if keyword.lower() in text_lower:
13                 logger.info(f"Proveedor identificado: {provider_id} (keyword: {keyword})")
14                 return provider_id
15
16     logger.info("Proveedor no identificado, usando configuración default")
17     return 'default'
```

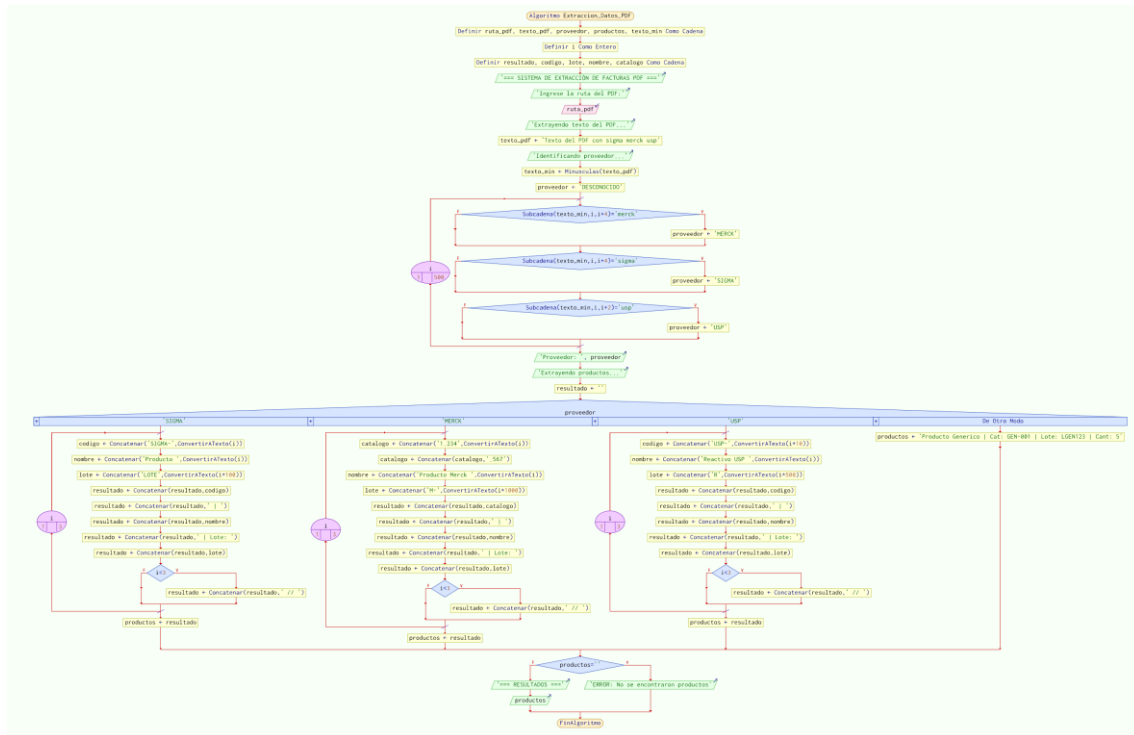
Extracción según el proveedor

```

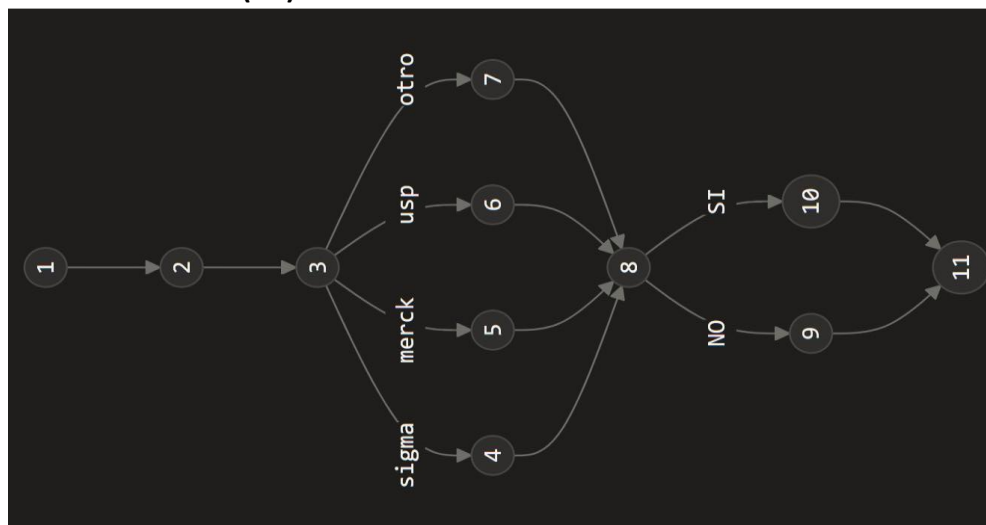
1  # *** EXTRACCIÓN ESPECÍFICA PARA SIGMA ***
2  if self.provider_id == 'sigma':
3      logger.info("Detectado proveedor SIGMA - usando extracción especializada")
4
5      # Extraer múltiples productos directamente del PDF usando tablas
6      products = self.extract_sigma_products_from_pdf(pdf_path)
7
8      # Extraer número de factura (común para todos los productos)
9      invoice = self.extract_by_anchor(self.raw_text, self.provider_config['anchors']['invoice'])
10     if not invoice:
11         invoice = self.fuzzy_search(self.raw_text, 'invoice',
12                                     ['invoice', 'factura', 'order', 'po'])
13
14     # Si hay productos, devolver lista completa
15     if len(products) > 0:
16         logger.info(f"Productos Sigma encontrados: {len(products)}")
17         return {
18             'company': 'SIGMA',
19             'invoice_number': invoice,
20             'products': products,
21             'multiple_items': True # Siempre True para que se guarde el array
22         }
23
24     # Si no se encontraron productos, continuar con extracción estándar
25     else:
26         logger.warning("No se encontraron productos con método especializado Sigma, usando método estándar")
27
28 # *** EXTRACCIÓN ESPECÍFICA PARA USP ***
29 if self.provider_id == 'usp':
30     logger.info("Detectado proveedor USP - usando extracción especializada")
31
32     # Extraer múltiples productos directamente del PDF usando tablas
33     products = self.extract_usp_products_from_pdf(pdf_path)
34
35     # Extraer número de factura (común para todos los productos)
36     invoice = self.extract_by_anchor(self.raw_text, self.provider_config['anchors']['invoice'])
37     if not invoice:
38         invoice = self.fuzzy_search(self.raw_text, 'invoice',
39                                     ['invoice', 'factura', 'order', 'po'])
40
41     # Si hay productos, devolver lista completa
42     if len(products) > 0:
43         logger.info(f"Productos USP encontrados: {len(products)}")
44         return {
45             'company': 'USP',
46             'invoice_number': invoice,
47             'products': products,
48             'multiple_items': True # Siempre True para que se guarde el array
49         }
50
51     # Si no se encontraron productos, continuar con extracción estándar
52     else:
53         logger.warning("No se encontraron productos con método especializado USP, usando método estándar")
54
55 # *** EXTRACCIÓN ESPECÍFICA PARA MERCK ***
56 if self.provider_id == 'merck':
57     logger.info("Detectado proveedor MERCK - usando extracción especializada")
58
59     # Extraer múltiples productos de la tabla
60     products = self.extract_merck_products_from_table(self.raw_text)
61
62     # Extraer número de factura (común para todos los productos)
63     invoice = self.extract_by_anchor(self.raw_text, self.provider_config['anchors']['invoice'])
64     if not invoice:
65         invoice = self.fuzzy_search(self.raw_text, 'invoice',
66                                     ['invoice', 'factura', 'order', 'po'])
67
68     # Si hay múltiples productos, devolver lista completa
69     if len(products) > 1:
70         logger.info(f"Múltiples productos encontrados: {len(products)}")
71         return {
72             'company': 'MERCK',
73             'invoice_number': invoice,
74             'products': products,
75             'multiple_items': True
76         }
77
78     # Si solo hay un producto, devolver en formato simple
79     elif len(products) == 1:
80         extracted = products[0].copy()
81         extracted['company'] = 'MERCK'
82         extracted['invoice_number'] = invoice
83         extracted['multiple_items'] = False
84         return extracted
85
86     # Si no se encontraron productos, continuar con extracción estándar
87     else:
88         logger.warning("No se encontraron productos con método especializado Merck, usando método estándar")
89
90 # *** EXTRACCIÓN ESTÁNDAR (para otros proveedores o fallback) ***
91 extracted = {}
92
93 # Número de lote
94 batch = self.extract_by_anchor(self.raw_text, self.provider_config['anchors']['batch'])
95 if not batch:
96     batch = self.fuzzy_search(self.raw_text, 'batch',
97                               ['batch', 'lote', 'lot', 'no. lote'])
98 extracted['batch_number'] = batch
99
100 # Nombre de producto
101 product = self.extract_product_name(self.raw_text, self.provider_config['anchors']['product'])
102 if not product:
103     product = self.fuzzy_search(self.raw_text, 'product',
104                                 ['product', 'producto', 'item', 'material'])
105 extracted['product_name'] = product
106
107 # Número de catálogo
108 catalog = self.extract_by_anchor(self.raw_text, self.provider_config['anchors']['catalog'])
109 if not catalog:
110     catalog = self.fuzzy_search(self.raw_text, 'catalog',
111                                 ['cat', 'catalog', 'sku', 'code'])
112 extracted['catalog_number'] = catalog
113
114 # Fecha de vencimiento
115 expiry = self.extract_date(self.raw_text, self.provider_config['anchors']['expiry'])
116 extracted['expiry_date'] = expiry
117
118 # Cantidad
119 quantity = self.extract_by_anchor(self.raw_text, self.provider_config['anchors']['quantity'])
120 extracted['quantity'] = quantity
121
122 # Número de factura
123 invoice = self.extract_by_anchor(self.raw_text, self.provider_config['anchors']['invoice'])
124 if not invoice:
125     invoice = self.fuzzy_search(self.raw_text, 'invoice',
126                                 ['invoice', 'factura', 'order', 'po'])
127 extracted['invoice_number'] = invoice
128
129 # Empresa/Proveedor
130 extracted['company'] = self.provider_id.upper() if self.provider_id != 'default' else None
131
132 # Limpiar valores None
133 extracted = {k: v for k, v in extracted.items() if v is not None}
134 extracted['multiple_items'] = False
135
136 logger.info(f"Extracción completada: {len(extracted)} campos encontrados")
137 logger.debug(f"Datos extraídos: {extracted}")
138
139 return extracted
140
141

```

3. DIAGRAMA DE FLUJO (DF)



4. GRAFO DE FLUJO (GF)



5. IDENTIFICACIÓN DE LAS RUTAS (Camino básico)

Camino Base 1 (SIGMA con productos):

1 → 2 → 3 → 4 → 8 → 10 → 11

Camino Base 2 (MERCK con productos):

1 → 2 → 3 → 5 → 8 → 10 → 11

Camino Base 3 (USP con productos):

$1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 10 \rightarrow 11$

Camino Base 4 (Otro Modo con productos):

$1 \rightarrow 2 \rightarrow 3 \rightarrow 7 \rightarrow 8 \rightarrow 10 \rightarrow 11$

Camino Base 5 (Cualquier proveedor SIN productos):

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 8 \rightarrow 9 \rightarrow 11$

Difiere de todos los anteriores en el nodo 8 (toma la rama NO en lugar de SI).

6. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

Nodos (N): Son todos los círculos numerados.

$N = 11$

Nodos predicados (P):

Entonces:

$P = 4$

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

$V(G) = 4 + 1$

$V(G) = 4 + 1$

$V(G) = 5$

➤ $V(G) = A - N + 2$

$V(G) = 14 - 11 + 2$

$V(G) = 5$

DONDE:

P: Número de nodos predicado

A: Número de aristas

N: Número de nodos