1. REQ005--- Detección de datos

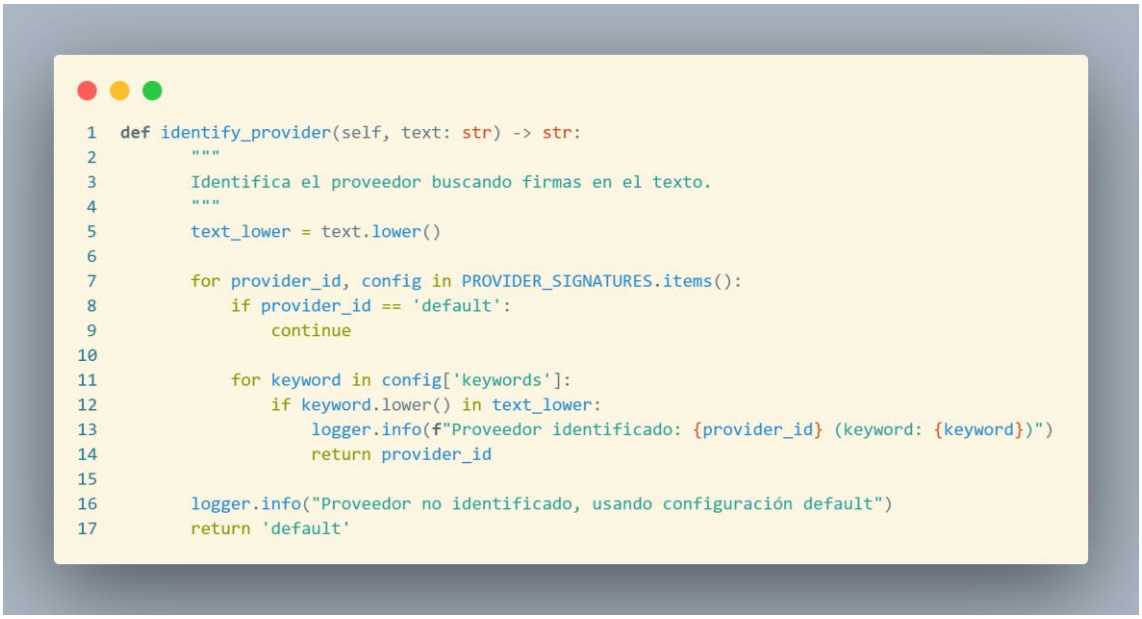
El sistema deberá tener una interfaz gráfica con acceso por credenciales únicas, El usuario podrá crear un usuario y contraseña, además deberá existir un usuario administrador que pueda restringir accesos y funciones del sistema, la validación de estos tendrá un límite de intento para preservar la seguridad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

2. CÓDIGO FUENTE

Extracción de texto

```python
def extract_text_from_pdf(self, pdf_path: str) -> str:
    """
    Extrae texto de un PDF usando PyPDF2 o pdfplumber.
    """
    try:
        # Intentar con PyPDF2 primero
        try:
            import PyPDF2
            with open(pdf_path, 'rb') as file:
                reader = PyPDF2.PdfReader(file)
                text = ""
                for page in reader.pages:
                    text += page.extract_text() + "\n"
                if text.strip():
                    return text
        except Exception as e:
            logger.warning(f"PyPDF2 falló: {e}")

        # Fallback a pdfplumber
        try:
            import pdfplumber
            text = ""
            with pdfplumber.open(pdf_path) as pdf:
                for page in pdf.pages:
                    page_text = page.extract_text()
                    if page_text:
                        text += page_text + "\n"

                    # Para Merck, también extraer tablas
                    if self.provider_id == 'merck':
                        tables = page.extract_tables()
                        if tables:
                            for table in tables:
                                for row in table:
                                    if row:
                                        text += " | ".join([str(cell) if cell else "" for cell in row]) + "\n"
            return text
        except Exception as e:
            logger.warning(f"pdfplumber falló: {e}")

    except Exception as e:
        logger.error(f"Error extrayendo texto del PDF: {e}")
        return ""

    return ""
```

Identificación del proveedor

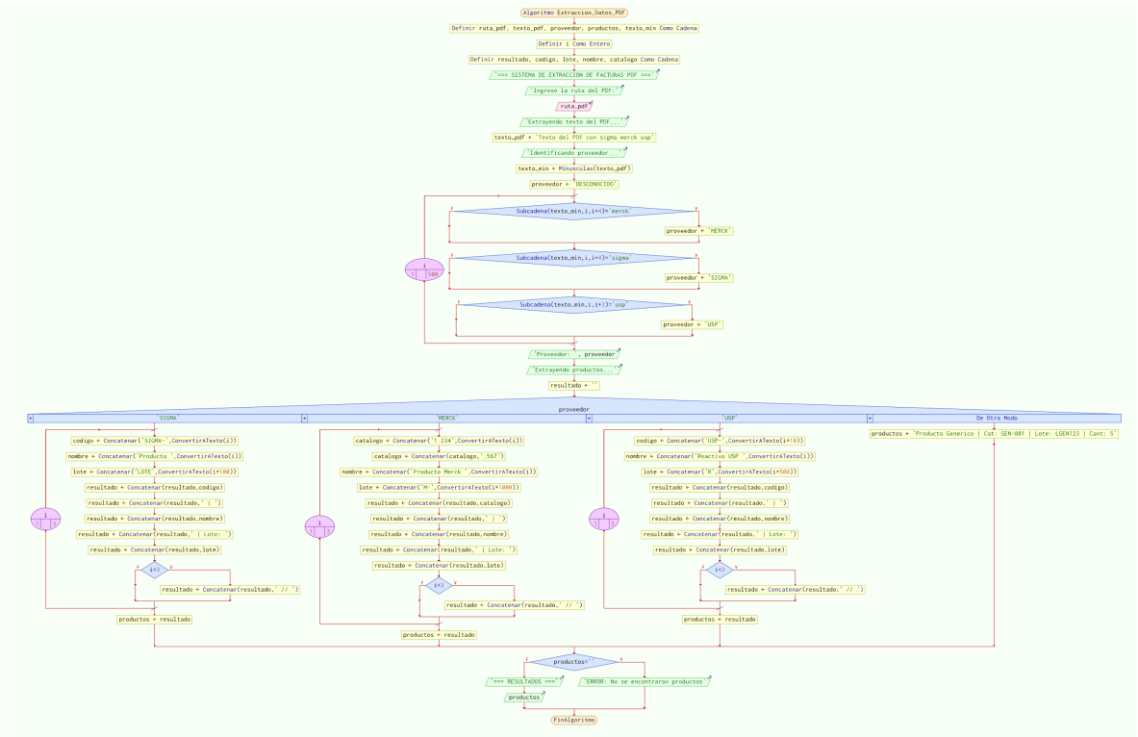


```python
def identify_provider(self, text: str) -> str:
        """
        Identifica el proveedor buscando firmas en el texto.
        """
        text_lower = text.lower()

        for provider_id, config in PROVIDER_SIGNATURES.items():
            if provider_id == 'default':
                continue

            for keyword in config['keywords']:
                if keyword.lower() in text_lower:
                    logger.info(f"Proveedor identificado: {provider_id} (keyword: {keyword})")
                    return provider_id

        logger.info("Proveedor no identificado, usando configuración default")
        return 'default'
```

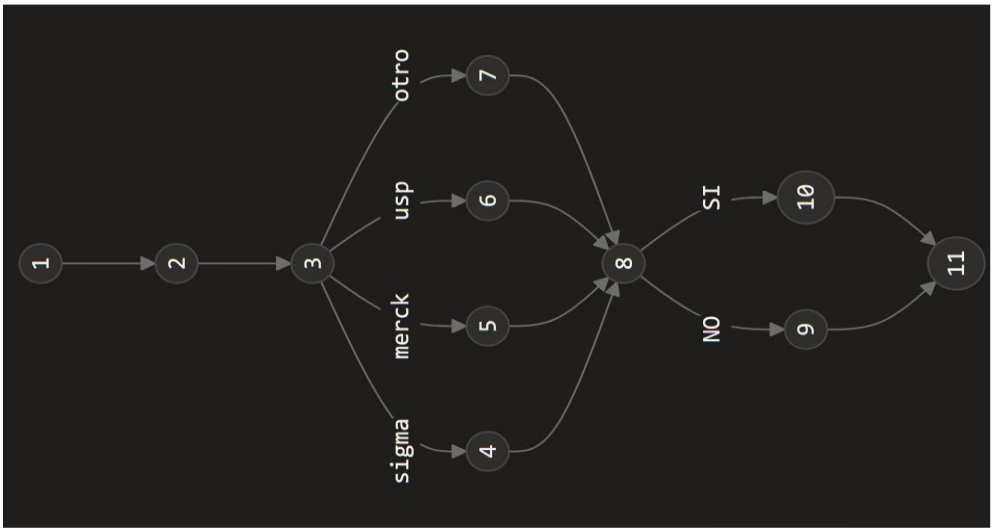
Extracción según el proveedor

```python
# *** EXTRACCIÓN ESPECÍFICA PARA SIGMA ***
    if self.provider_id == 'sigma':
        logger.info("Detectado proveedor SIGMA - usando extracción especializada")

        # Extraer múltiples productos directamente del PDF usando tablas
        products = self.extract_sigma_products_from_pdf(pdf_path)

        # Extraer número de factura (común para todos los productos)
        invoice = self.extract_by_anchor(self.raw_text, self.provider_config['anchors']['invoice'])
        if not invoice:
            invoice = self.fuzzy_search(self.raw_text, 'invoice',
                                        ['invoice', 'factura', 'order', 'po'])

        # Si hay productos, devolver lista completa
        if len(products) > 0:
            logger.info(f"Productos Sigma encontrados: {len(products)}")
            return {
                'company': 'SIGMA',
                'invoice_number': invoice,
                'products': products,
                'multiple_items': True  # Siempre True para que se guarde el array
            }

        # Si no se encontraron productos, continuar con extracción estándar
        else:
            logger.warning("No se encontraron productos con método especializado Sigma, usando método estándar")

    # *** EXTRACCIÓN ESPECÍFICA PARA USP ***
    if self.provider_id == 'usp':
        logger.info("Detectado proveedor USP - usando extracción especializada")

        # Extraer múltiples productos directamente del PDF usando tablas
        products = self.extract_usp_products_from_pdf(pdf_path)

        # Extraer número de factura (común para todos los productos)
        invoice = self.extract_by_anchor(self.raw_text, self.provider_config['anchors']['invoice'])
        if not invoice:
            invoice = self.fuzzy_search(self.raw_text, 'invoice',
                                        ['invoice', 'factura', 'order', 'po'])

        # Si hay productos, devolver lista completa
        if len(products) > 0:
            logger.info(f"Productos USP encontrados: {len(products)}")
            return {
                'company': 'USP',
                'invoice_number': invoice,
                'products': products,
                'multiple_items': True  # Siempre True para que se guarde el array
            }

        # Si no se encontraron productos, continuar con extracción estándar
        else:
            logger.warning("No se encontraron productos con método especializado USP, usando método estándar")

    # *** EXTRACCIÓN ESPECÍFICA PARA MERCK ***
    if self.provider_id == 'merck':
        logger.info("Detectado proveedor MERCK - usando extracción especializada")

        # Extraer múltiples productos de la tabla
        products = self.extract_merck_products_from_table(self.raw_text)

        # Extraer número de factura (común para todos los productos)
        invoice = self.extract_by_anchor(self.raw_text, self.provider_config['anchors']['invoice'])
        if not invoice:
            invoice = self.fuzzy_search(self.raw_text, 'invoice',
                                        ['invoice', 'factura', 'order', 'po'])

        # Si hay múltiples productos, devolver lista completa
        if len(products) > 1:
            logger.info(f"Múltiples productos encontrados: {len(products)}")
            return {
                'company': 'MERCK',
                'invoice_number': invoice,
                'products': products,
                'multiple_items': True
            }

        # Si solo hay un producto, devolver en formato simple
        elif len(products) == 1:
            extracted = products[0].copy()
            extracted['company'] = 'MERCK'
            extracted['invoice_number'] = invoice
            extracted['multiple_items'] = False
            return extracted

        # Si no se encontraron productos, continuar con extracción estándar
        else:
            logger.warning("No se encontraron productos con método especializado Merck, usando método estándar")

    # *** EXTRACCIÓN ESTÁNDAR (para otros proveedores o fallback) ***
    extracted = {}

    # Número de lote
    batch = self.extract_by_anchor(self.raw_text, self.provider_config['anchors']['batch'])
    if not batch:
        batch = self.fuzzy_search(self.raw_text, 'batch',
                                  ['batch', 'lote', 'lot', 'no. lote'])
    extracted['batch_number'] = batch

    # Nombre de producto
    product = self.extract_product_name(self.raw_text, self.provider_config['anchors']['product'])
    if not product:
        product = self.fuzzy_search(self.raw_text, 'product',
                                    ['product', 'producto', 'item', 'material'])
    extracted['product_name'] = product

    # Número de catálogo
    catalog = self.extract_by_anchor(self.raw_text, self.provider_config['anchors']['catalog'])
    if not catalog:
        catalog = self.fuzzy_search(self.raw_text, 'catalog',
                                    ['cat', 'catalog', 'sku', 'code'])
    extracted['catalog_number'] = catalog

    # Fecha de vencimiento
    expiry = self.extract_date(self.raw_text, self.provider_config['anchors']['expiry'])
    extracted['expiry_date'] = expiry

    # Cantidad
    quantity = self.extract_by_anchor(self.raw_text, self.provider_config['anchors']['quantity'])
    extracted['quantity'] = quantity

    # Número de factura
    invoice = self.extract_by_anchor(self.raw_text, self.provider_config['anchors']['invoice'])
    if not invoice:
        invoice = self.fuzzy_search(self.raw_text, 'invoice',
                                    ['invoice', 'factura', 'order', 'po'])
    extracted['invoice_number'] = invoice

    # Empresa/Proveedor
    extracted['company'] = self.provider_id.upper() if self.provider_id != 'default' else None

    # Limpiar valores None
    extracted = {k: v for k, v in extracted.items() if v is not None}
    extracted['multiple_items'] = False

    logger.info(f"Extracción completada: {len(extracted)} campos encontrados")
    logger.debug(f"Datos extraídos: {extracted}")

    return extracted
```

3. DIAGRAMA DE FLUJO (DF)



4. GRAFO DE FLUJO (GF)



5. IDENTIFIACCIÒN 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 → 2 → 3 → 6 → 8 → 10 → 11

Camino Base 4 (Otro Modo con productos):

1 → 2 → 3 → 7 → 8 → 10 → 11

Camino Base 5 (Cualquier proveedor SIN productos):

1 → 2 → 3 → 4 → 8 → 9 → 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) = 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