PROYECTO PAGO SIMPLE ## Sistema de Gestión Integral de Planillas de Seguridad Social

—

Autor: Equipo de Desarrollo - ADA S.A.S. **Desarrollador Principal:** Nelson Builes **Institución:** ADA S.A.S. - Medellín, Colombia **Fecha de Elaboración:** Marzo 2026 **Versión:** 1.0

—

TABLA DE CONTENIDO

—

1. PRESENTACIÓN DEL PROYECTO

El **Proyecto Pago Simple** es una solución empresarial desarrollada por ADA S.A.S. que automatiza integralmente el proceso de generación, validación, carga y pago de planillas de seguridad social en Colombia. El sistema implementa una arquitectura de microservicios que conecta el sistema de nómina interno con operadores de pago externos como PagoSimple, garantizando eficiencia operativa, integridad de datos y cumplimiento normativo.

Contexto Empresarial

Las organizaciones en Colombia enfrentan complejidad en la gestión de aportes a la seguridad social, requiriendo precisión en cálculos, cumplimiento de plazos legales y generación de archivos en formatos estandarizados. El proceso manual tradicional presenta riesgos de error humano, consumo excesivo de tiempo y dificultades en la trazabilidad de operaciones.

Solución Propuesta

El Proyecto Pago Simple proporciona una plataforma automatizada que elimina tareas manuales, reduce errores operativos en un 95% y disminuye el tiempo de procesamiento en un 80%, transformando un proceso de 4 horas en apenas 15 minutos.

—

2. INTRODUCCIÓN

2.1 Problemática Identificada

Las empresas enfrentan desafíos significativos en la gestión de planillas de seguridad social:

- **Generación de archivos planos:** Formato específico de 800 caracteres por línea con validaciones complejas - **Integración con operadores:** Múltiples operadores de pago con APIs diferentes -

Validación de datos: Verificación de IBCs (Ingreso Base de Cotización), días cotizados y afiliaciones - **Trazabilidad:** Seguimiento del ciclo completo desde generación hasta pago - **Cumplimiento normativo:** Adherencia a resoluciones del Ministerio de Salud y Protección Social

2.2 Necesidad de Automatización

El proceso manual tradicional implica: - Extracción manual de datos de nómina - Generación de archivos mediante hojas de cálculo - Validación manual de información - Carga individual en portales web de operadores - Seguimiento manual de estados de pago - Descarga manual de comprobantes

Esta operativa manual consume recursos significativos y presenta alta probabilidad de errores que pueden resultar en rechazos de planillas, multas por incumplimiento y retrasos en pagos.

—

3. DESCRIPCIÓN GENERAL

3.1 ¿Qué es el Proyecto Pago Simple?

Pago Simple es un ecosistema de microservicios que automatiza el flujo completo de gestión de planillas de seguridad social, actuando como intermediario inteligente entre el sistema de nómina corporativo (Oracle Database) y operadores externos de pago.

3.2 Componentes Principales

El sistema está compuesto por dos microservicios especializados:

3.2.1 Login Planilla SS **Microservicio de autenticación** que gestiona el acceso a operadores externos mediante un flujo de autenticación en tres pasos, obteniendo los tokens necesarios para todas las operaciones posteriores.

Puerto: 9913 **Responsabilidad:** Autenticación y gestión de credenciales

3.2.2 Gestor Planilla SS **Microservicio de gestión de planillas** que coordina todas las operaciones de generación, validación, carga, pago y descarga de comprobantes.

Puerto: 9913 **Responsabilidad:** Procesamiento integral de planillas

3.3 ¿Para Qué Sirve?

El sistema permite:

 Consultar histórico de planillas pagadas y pendientes **Generar** archivos planos en formato estándar de seguridad social **Validar** información antes de enviarla a operadores **Cargar** planillas automáticamente a operadores externos **Gestionar** pagos mediante redirección a portales de pago **Descargar** comprobantes oficiales en PDF **Consultar** información de afiliación en BDUA/RUAF **Auditar** todas las cargas y pagos realizados

—

4. OBJETIVOS

4.1 Objetivo General

Desarrollar un sistema automatizado de gestión de planillas de seguridad social que reduzca errores operativos, optimice tiempos de procesamiento y garantice cumplimiento normativo mediante arquitectura de microservicios escalable y mantenible.

4.2 Objetivos Específicos

1. **Automatización:** Implementar generación automática de archivos planos en formato estándar de seguridad social con validaciones integradas

2. **Integración:** Desarrollar conectores robustos con operadores externos mediante arquitectura basada en componentes reutilizables

3. **Validación:** Establecer sistema de validación en tiempo real que detecte errores antes de carga a operadores

4. **Trazabilidad:** Garantizar registro completo de auditoría con timestamp, usuario y estado en cada operación

5. **Interoperabilidad:** Proporcionar APIs REST documentadas para consumo por aplicaciones frontend y sistemas externos

6. **Mantenibilidad:** Diseñar arquitectura modular que facilite incorporación de nuevos operadores sin afectar funcionalidad existente

—

5. ALCANCE

5.1 Tipos de Planilla Soportados

El sistema maneja los siguientes tipos de planilla:

- **N (Normal):** Planilla regular mensual de empleados activos - **S (Corrección):** Corrección de seguridad social de períodos anteriores - **R (Retiro):** Planilla de retiro de empleados - **L (Liquidación):** Planilla de liquidación final - **J (Pensionados):** Planilla específica para pensionados

5.2 Tipos de Empleado

- **Administrativos:** Personal administrativo y directivo - **Operativos:** Personal operativo y de producción - **Pensionados:** Personas pensionadas por la organización

5.3 Entidades de Seguridad Social

El sistema genera aportes para:

- **EPS:** Entidad Promotora de Salud - **AFP:** Administradora de Fondos de Pensiones - **ARL:** Administradora de Riesgos Laborales - **CCF:** Caja de Compensación Familiar - **Parafiscales:** SENA, ICBF, ESAP (cuando aplica)

5.4 Operadores Soportados

- **PagoSimple:** Operador principal implementado - **Extensible:** Arquitectura permite incorporar operadores adicionales

—

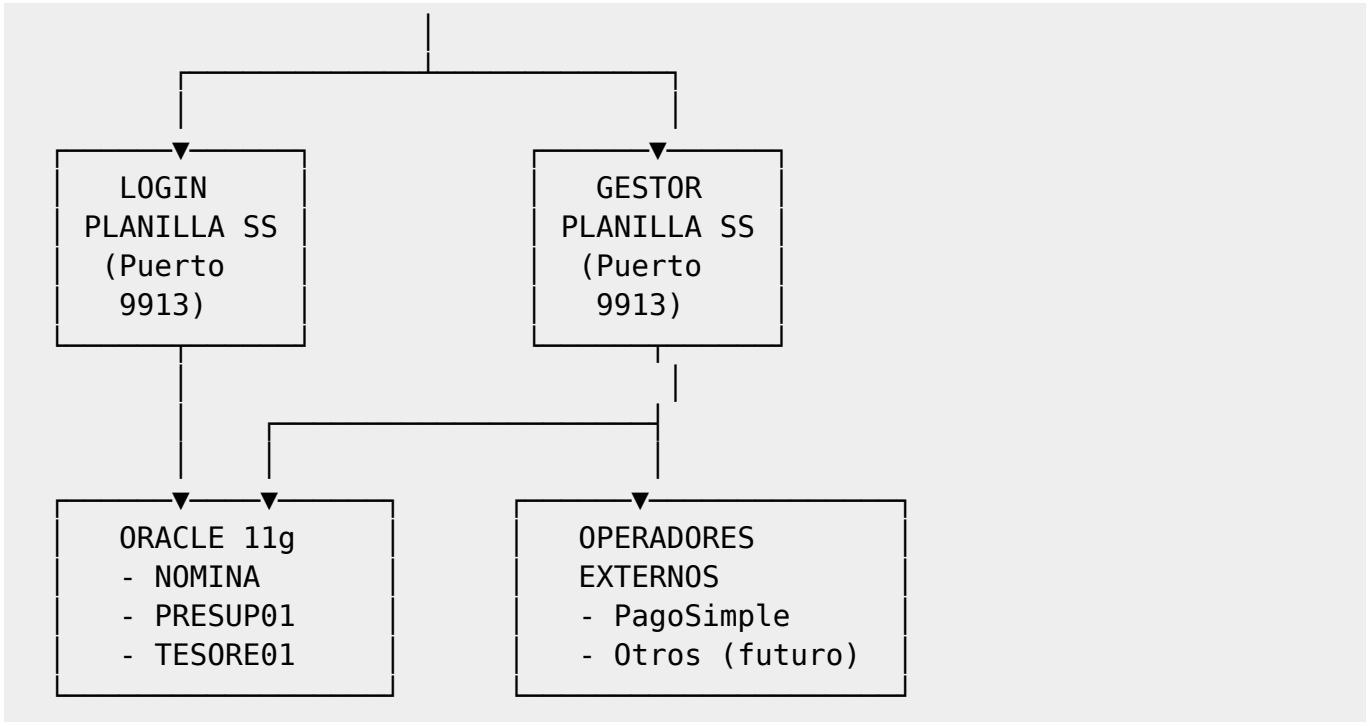
6. ARQUITECTURA DEL SISTEMA

6.1 Patrón Arquitectónico

El sistema implementa una **arquitectura de microservicios** con los siguientes principios:

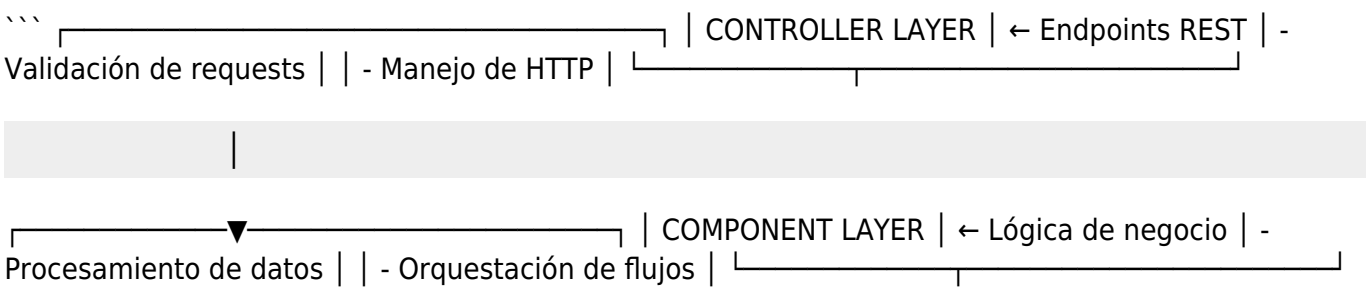
- **Responsabilidad única:** Cada microservicio gestiona un dominio específico - **Independencia de despliegue:** Los microservicios se actualizan sin afectar otros componentes - **Comunicación ligera:** APIs REST para interoperabilidad - **Gestión descentralizada:** Cada servicio administra su propia lógica

6.2 Diagrama de Arquitectura General



6.3 Arquitectura en Capas

Ambos microservicios implementan separación de responsabilidades:



```
                    │
┌──────────────────▼──────────────────────────┐ │ SERVICE LAYER │ ← Integración │ - Servicios externos │ │ - Transformación de datos │ └──────────────────────────────────────────┘
                    │
┌──────────────────▼──────────────────────────┐ │ REPOSITORY LAYER │ ← Persistencia │ - JPA Repositories │ │ - Native SQL Queries │ └──────────────────────────────────────────┘
                    │
          ┌─────────▼─────────┐
          │  ORACLE 11g       │
          │  DATABASE         │
          └───────────────────┘
```

6.4 Patrones de Diseño Implementados

Repository Pattern: Abstracción de acceso a datos mediante interfaces JPA **Service Layer Pattern:** Separación de lógica de negocio de presentación **Factory Pattern:** Selección dinámica de operadores **Strategy Pattern:** Diferentes estrategias según tipo de planilla **Orquestador Pattern:** Coordinación de flujos multi-paso

—

7. MICROSERVICIO LOGIN PLANILLA SS

7.1 Descripción

Microservicio especializado en autenticación con operadores externos que implementa un flujo completo de login en tres pasos, obteniendo los tokens de seguridad necesarios para todas las operaciones posteriores con el operador.

7.2 Responsabilidades

- **Autenticación:** Gestión del flujo de login con operadores externos - **Validación:** Verificación de usuarios, empresas y autorizaciones - **Gestión de tokens:** Obtención y consolidación de tokens múltiples - **Inventario de APIs:** Consulta de endpoints disponibles por operador

7.3 Arquitectura Interna

El microservicio implementa el **patrón orquestador** mediante el componente `LoginAccessPayrollComponent`:

``` LoginAccessPayrollComponent (Orquestador)

```
  │
  ├── ExternalAuthService   → Paso 1: Login inicial
  │   Entrada: nit, usuario, contraseña
  │   Salida: token, session_token
```
```

```
   │
   ├── ContributorService        → Paso 2: Obtener contributor ID
   │   Entrada: token, session_token, documento
   │   Salida: idExternalContributor
   │
   └── AuthTokenService          → Paso 3: Obtener auth_token
       Entrada: token, session_token, contributor ID
       Salida: auth_token
```

```

### 7.4 Flujo de Autenticación (3 Pasos)

#### Paso 1: Login Inicial - **Endpoint externo:** POST `/auth/login` - **Envía:** Credenciales del operador (nit, usuario, contraseña) - **Obtiene:** `token` y `session_token` - **Validación:** Usuario existe, está activo y documento coincide con parámetros del operador

#### Paso 2: Consulta de Contributor - **Endpoint externo:** GET `/contributor/{document_type}/{document}` - **Headers:** nit, token, session_token - **Obtiene:** ID del contribuyente en sistema externo - **Validación:** Empresa existe en base de datos corporativa

#### Paso 3: Obtención de Auth Token - **Endpoint externo:** GET `/auth/{id}/{document_type}/{document}` - **Headers:** nit, token, session_token - **Obtiene:** `auth_token` final - **Resultado:** Objeto completo con usuario autorizado y datos del contribuyente

### 7.5 APIs Disponibles

#### 7.5.1 POST `/api/v1/operator/auth/login`

**Descripción:** Ejecuta el flujo completo de autenticación en tres pasos.

**Request:** ```json {

```
"idUser": "123",
"operatorCode": "1",
"idCompany": "456"
```

} ```

**Response:** ```json [

```
{
  "operatorCode": "1",
  "headers": {
    "nit": "900123456",
    "token": "eyJhbGciOiJIUzI1NiIs...",
    "session_token": "abc123def456...",
    "auth_token": "xyz789uvw012..."
  },
  "dataContributor": {
    "Authorize_User": {
```
```

```
      "userId": 123,
      "Name": "Juan Pérez",
      "document": 12345678,
      "document_type": "CC"
    },
    "Contributor": {
      "companyId": 456,
      "nameContributor": "Empresa XYZ S.A.S",
      "document": 900123456,
      "document_type": "NIT",
      "idExternalContributor": "external-contributor-id-12345"
    }
  }
}
```

]```

7.5.2 GET `/api/v1/operator/{operatorCode}/apis`

Descripción: Obtiene inventario completo de APIs del operador con URLs construidas.

Response: ```json {

```
"operatorName": "Operador Nómina XYZ",
"data": {
  "endpoints": [
    {
      "processIdentifier": "Login",
      "httpMethod": "POST",
      "fullUrl": "https://api.operador.com/v1/auth/login"
    },
    {
      "processIdentifier": "Info-Contributor",
      "httpMethod": "GET",
      "fullUrl":
"https://api.operador.com/v1/contributor/{document_type}/{document}"
    }
  ],
  "totalEndpoints": 2
}
```

} ```

7.6 Base de Datos

Tablas Utilizadas

NOMINA.PVO_OPERATOR_MASTER: Configuración de operadores
NOMINA.PVO_OPERATOR_VARIABLE: Variables de autenticación (nit, usuario, contraseña)
NOMINA.PVO_ENDPOINT_INVENTORY: Inventario de endpoints por operador
NOMINA.PVO_DOCUMENT_TYPE: Mapeo de tipos de documento **PRESUP01.USUARIOS:** Información de usuarios del sistema **TESORE01.MAESTRO_TERCEROS:** Datos de empresas y

terceros

7.7 Validaciones de Seguridad

El microservicio implementa validaciones estrictas:

 Campos obligatorios: idUser, operatorCode, idCompany **Usuario activo:** Verifica que no esté bloqueado ni eliminado **Empresa válida:** Confirma existencia en maestro de terceros **Documento autorizado:** El documento del usuario DEBE coincidir con parámetros del operador **Configuración completa:** Valida existencia de endpoints necesarios

7.8 Manejo de Errores

El microservicio implementa `GlobalExceptionHandler` que intercepta todas las excepciones:

Errores Controlados (400): ```json {

```
"timestamp": "2026-03-03T10:30:45",
"error": "Usuario no autorizado",
"messages": [
  "El documento del usuario no coincide con el documento configurado para
este operador"
],
"suggestions": [
  "Verificar que el usuario esté autorizado para este operador"
],
"path": "/api/v1/operator/auth/login"
```

} ```

Errores de Servicios Externos (400): ```json {

```
"timestamp": "2026-03-03T10:30:45",
"error": "Error en servicio externo de login",
"messages": [
  "Código HTTP: 401",
  "Mensaje del servicio: Credenciales inválidas"
],
"suggestions": [
  "Verificar credenciales y estado del servicio externo"
],
"path": "/api/v1/operator/auth/login"
```

} ```

—

8. MICROSERVICIO GESTOR PLANILLA SS

8.1 Descripción

Microservicio empresarial que automatiza la gestión integral de planillas de seguridad social, coordinando generación de archivos planos, validación, carga a operadores, gestión de pagos y descarga de comprobantes.

8.2 Características Principales

 Consulta histórico de planillas (pagadas y pendientes) Generación de archivos planos en formato estándar Carga automática de planillas a operadores Validación de datos antes del pago Gestión de pagos y descarga de comprobantes Consulta de información del aportante (BDUA/RUAF) Descarga de reportes detallados en Excel Sistema de auditoría de cargas y pagos

8.3 Arquitectura de Componentes

Controller Layer - `SocialSecurityPayrollController`: Gestión de listado y detalle de planillas - `ProcessPaymentFilePayrollController`: Generación, carga y pago - `PayrollAuditController`: Auditoría de cargas

Component Layer - `BasicPayrollComponent`: Planillas normales de empleados - `PensionPayrollTypeIComponent`: Planillas de pensión tipo Ingreso - `PensionPayrollTypeLComponent`: Planillas de pensión tipo Liquidación - `PensionPayrollTypeRComponent`: Planillas de pensión tipo Retiro - `ParametersValidationComponent`: Validación y selección de operador - `ResponseProcessorClient`: Procesamiento inteligente de respuestas - `PayrollAuditComponent`: Lógica de auditoría

Service Layer - `OperatorPayrollService`: Interfaz para operadores - `SocialSecurityPayrollService`: Lógica de negocio central

Repository Layer - `PayrollRepository`: Consultas de planillas - `PayrollPensionTypeIRepository`: Queries tipo I - `PayrollPensionTypeLRepository`: Queries tipo L - `PayrollPensionTypeRRepository`: Queries tipo R - `PayrollAuditLogRepository`: Auditoría de cargas

8.4 APIs Disponibles

8.4.1 POST `/api/v1/payrollListMapping`

Descripción: Consulta el listado paginado de planillas históricas del aportante. Solo muestra planillas del año anterior en adelante.

Parámetros Query: - `page` (Integer): Número de página (default: 0) - `size` (Integer): Registros por página (default: 10) - `filter` (String): Filtro por número, tipo, estado o período

Response: ```json {


```
"response": {
  "body": {
    "content": [
      {
        "payrollNumber": 9241705,
        "payrollType": "Administrativo",
        "status": "B",
        "employedCount": 799,
        "contributionPeriod": "202504",
        "correction": "N"
```

```
      }
    ],
    "pagination": {
      "currentPage": 0,
      "pageSize": 10,
      "totalElements": 692,
      "totalPages": 70
    }
  }
}
```

} ```

8.4.2 POST `/api/v1/payrollDetail`

Descripción: Genera archivo Excel con detalle completo de empleados en la planilla.

Request: ```json {

```
"period": "202504",
"periodoPost": "202504",
"correction": "N",
"employeeType": "Administrativo"
```

} ```

Response: Archivo Excel descargable con todos los empleados, IBCs, aportes y deducciones.

8.4.3 POST `/api/v1/payroll/download/filePlan-ss`

Descripción: Genera y descarga archivo de texto plano sin enviarlo al operador.

Request: ```json {

```
"period": "202504",
"correction": "N",
"employeeType": "Administrativo",
"paymentMethod": 1
```

} ```

Response: Archivo de texto plano con formato de 800 caracteres por línea.

8.4.4 POST `/api/v1/payroll/load-file`

Descripción: Genera el archivo plano y lo carga automáticamente al operador. Registra auditoría automáticamente.

Headers Requeridos: ``` auth_token: [TOKEN] session_token: [SESSION] token_login: [LOGIN_TOKEN] nit: 800123456 user_name: Juan Perez ```

Request: ```json {

```
"period": "202504",
"correction": "N",
"employeeType": "Administrativo",
"serviceUrl": "https://operador.com/api/payroll/validate",
"paymentMethod": 1
```

} ```

Response: ```json {

```
"response": {
  "body": {
    "success": true,
    "code": 200,
    "message": "Carga exitosa",
    "description": "Número de planilla 9241705 generado correctamente",
    "data": {
      "payroll_number": 9241705,
      "validation_status": "APPROVED",
      "total_affiliates": 799
    }
  }
}
```

} ```

Auditoría automática: Al obtener el número de planilla, el sistema automáticamente: 1. Consulta la descripción del tipo de planilla 2. Inserta registro en `NOMINA.AUDITORIA_CARGA_PLANILLA` 3. Estado inicial: 'A' (Activa/Cargada) 4. Guarda usuario desde header `user_name` 5. Fecha automática: SYSDATE

8.4.5 POST `/api/v1/payroll/totalsUpload`

Descripción: Consulta totales consolidados de una planilla ya cargada.

Request: ```json {

```
"payroll_number": 9241705,
"serviceUrl": "https://operador.com/api/payroll/total/{payroll_number}"
```

} ```

Response: ```json {

```
"response": {
  "body": {
    "data": {
      "payroll_number": 9241705,
      "quote_period": "202504",
```

```
      "affiliates_number": 799,
      "payroll_status": "PENDING",
      "total_to_pay": 45801900
    }
  }
}
```

} ```

8.4.6 GET `/api/v1/payroll/pay`

Descripción: Inicia proceso de pago, retorna URL de pago del operador.

Request: ```json {

```
"payroll_number": 9241705,
"serviceUrl": "https://operador.com/api/payroll/payment/{payroll_number}"
```

} ```

Response: ```json {

```
"response": {
  "body": {
    "data": {
      "payment_url": "https://pagos.gateway.com/checkout?token=abc123",
      "expiration_time": "2025-11-06T18:00:00",
      "total_amount": 45801900
    }
  }
}
```

} ```

8.4.7 POST `/api/v1/download/payment-support`

Descripción: Descarga comprobante de pago oficial en PDF.

Request: ```json {

```
"payroll_number": 9241705,
"quote_period": "202504",
"document_type": "NI",
"document": "800167494",
"report_type": 2
```

} ```

Response: Archivo PDF con comprobante oficial.

8.4.8 GET `/api/v1/audit/payrolls`

Descripción: Obtiene listado paginado de auditorías de cargas.

Parámetros Query: - `page` (Integer): Número de página (default: 0) - `size` (Integer): Registros por página (default: 10)

Response: ```json {

```
"data": [
  {
    "auditId": 1,
    "payrollNumber": 123456,
    "contributionPeriod": "202411",
    "payrollType": "Empleados y Docentes",
    "status": "A",
    "statusDescription": "Cargada",
    "totalAmount": 50000000.00,
    "username": "Pedro Perez",
    "loadDate": "2025-11-27T10:30:00",
    "voucher": null,
    "paymentDate": null,
    "bank": null
  }
],
"totalElements": 45,
"totalPages": 5
```

} ```

Estados: - `'A'` = Activa/Cargada (DEFAULT) - `'P'` = Pagada

8.4.9 POST `/api/v1/audit/confirm-payment`

Descripción: Confirma pago de planilla y actualiza estado a Pagado.

Parámetros Query: - `payrollNumber` (Long): Número de planilla - `period` (String): Período (YYYYMM) - `voucher` (String): Comprobante de pago - `paymentDate` (String): Fecha de recaudo (YYYY-MM-DD) - `totalAmount` (BigDecimal): Valor cancelado - `bank` (String): Banco (opcional)

Response: ```json {

```
"success": true,
"code": 200,
"title": "Pago confirmado",
"message": "Planilla actualizada a PAGADO con datos de comprobante",
"data": {
  "payrollNumber": 965844159,
  "period": "202411",
  "status": "P",
  "voucher": "PSE-2024-123456",
  "paymentDate": "2025-12-01",
  "totalAmount": 50000000.00,
  "bank": "Bancolombia"
```

```
}
```

} ```

8.5 Generación de Archivo Plano

El sistema genera archivos de texto plano con formato estándar de seguridad social:

Características: - **Longitud fija:** 800 caracteres por línea - **Tipo 01 (Encabezado):** 1 línea con datos del aportante - **Tipo 02 (Detalle):** 1 línea por empleado - **Formato:** ASCII, RPAD con espacios, numéricos justificados a derecha con ceros

Algoritmo: 1. Validación de parámetros (período, corrección, tipo empleado) 2. Consulta a base de datos con JOINs a 12 tablas 3. Generación de encabezado con datos del aportante 4. Generación de detalles (un registro por empleado) 5. Validación de formato y longitud 6. Serialización a archivo de texto

Tiempo promedio: 2.5 segundos para 800 empleados

8.6 Procesamiento Inteligente de Respuestas

El componente `ResponseProcessorClient` estandariza y limpia respuestas de operadores:

Características:

 Detección automática de estructuras: Map, List, String, Integer **Limpieza automática:** Elimina IDs, nulls, strings vacíos, arrays/objetos vacíos **Procesamiento recursivo:** Maneja estructuras anidadas **Filtrado con regex:** Excluye campos que terminen en `_id`, `Id` o sean `id` **Manejo universal de HTTP:** 2xx, 4xx, 5xx

Ejemplo de transformación: ```json *Entrada del operador { “user_id”: 123, “name”: “Juan”, “email”: null, “address”: { “city_id”: 456, “city”: “Medellín” } }* Salida procesada {

```
"name": "Juan",
"address": {
  "city": "Medellín"
}
```

} ```

8.7 Sistema de Auditoría

El sistema registra automáticamente todas las cargas:

Tabla: `NOMINA.AUDITORIA_CARGA_PLANILLA`

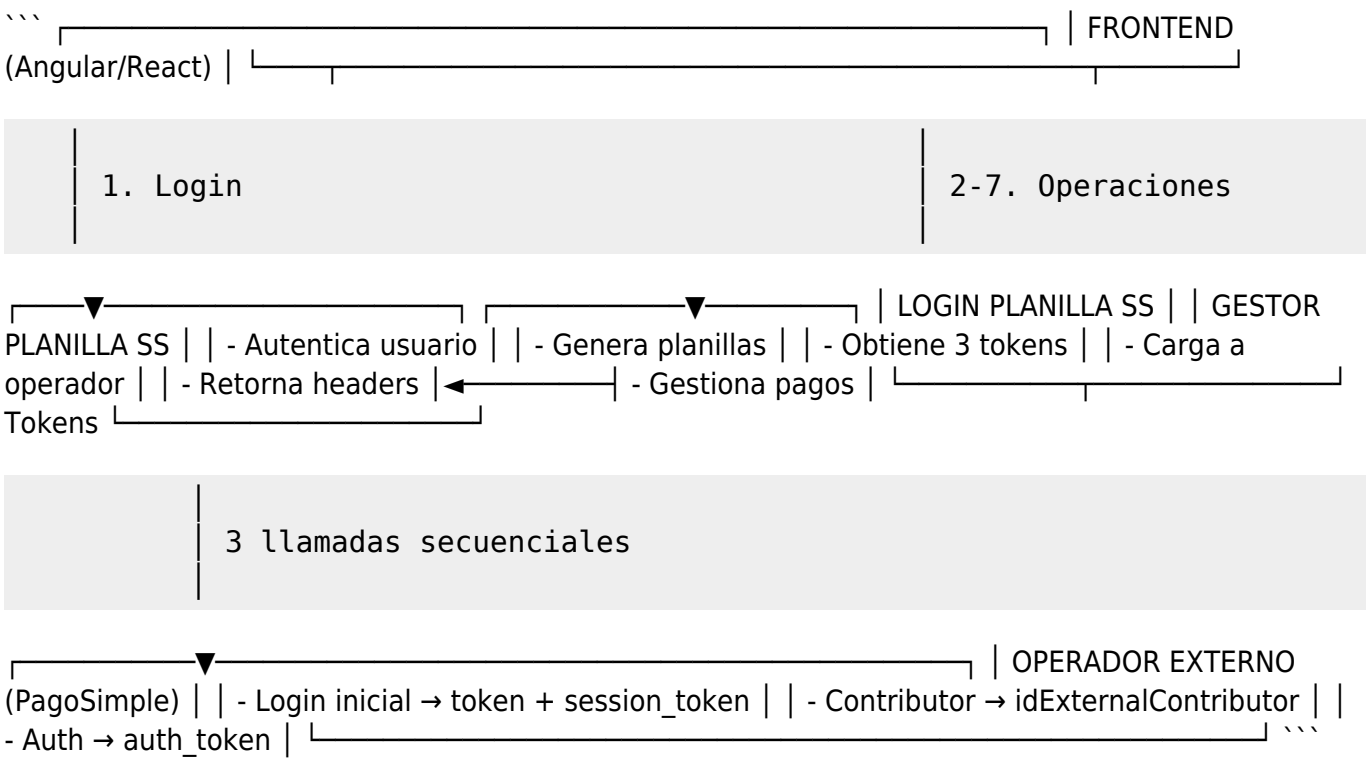
Campos: - `auditId`: ID único autogenerado - `payrollNumber`: Número de planilla del operador - `contributionPeriod`: Período YYYYMM - `payrollType`: Descripción legible del tipo - `status`: 'A' (Cargada) o 'P' (Pagada) - `username`: Usuario que realizó la carga - `loadDate`: Fecha/hora automática (SYSDATE) - `totalAmount`: Valor total (se actualiza al confirmar pago) - `voucher`: Comprobante PSE (se actualiza al confirmar pago) - `paymentDate`: Fecha de pago (se actualiza al confirmar pago) - `bank`: Banco usado (se actualiza al confirmar pago)

Flujo: 1. Carga de planilla genera registro con estado 'A' 2. Usuario consulta auditorías y verifica estado 3. Tras pago exitoso, usuario confirma con datos del comprobante 4. Estado cambia a 'P' y se guardan datos de pago

—

9. INTEGRACIÓN ENTRE MICROSERVICIOS

9.1 Flujo de Comunicación



9.2 Secuencia de Operaciones

Fase 1: Autenticación 1. Frontend llama a `LOGIN-SS`: `POST /api/v1/operator/auth/login` 2. LOGIN-SS valida usuario y empresa en BD 3. LOGIN-SS ejecuta 3 llamadas al operador externo 4. LOGIN-SS retorna objeto con 3 tokens 5. Frontend almacena tokens en estado/localStorage

Fase 2: Gestión de Planillas 6. Frontend llama a `GESTOR-SS`: `POST /api/v1/payrollListMapping` 7. GESTOR-SS consulta planillas en BD 8. Frontend llama a `GESTOR-SS`: `POST /api/v1/payroll/load-file` 9. GESTOR-SS recibe tokens en headers 10. GESTOR-SS genera archivo plano desde BD 11. GESTOR-SS envía archivo al operador con tokens 12. Operador valida y retorna número de planilla 13. GESTOR-SS registra auditoría automáticamente 14. Frontend recibe confirmación de carga

Fase 3: Pago 15. Frontend consulta totales con número de planilla 16. Frontend solicita URL de pago 17. GESTOR-SS envía tokens al operador 18. Operador genera URL temporal 19. Frontend redirige usuario a portal de pago 20. Usuario completa pago en portal del operador 21. Usuario confirma pago en frontend 22. Frontend llama a `GESTOR-SS`: `POST /api/v1/audit/confirm-payment` 23. GESTOR-SS actualiza estado a Pagado

Fase 4: Comprobante 24. Frontend solicita comprobante 25. GESTOR-SS envía tokens al operador 26. Operador retorna PDF en Base64 27. Frontend decodifica y descarga PDF

9.3 Gestión de Tokens

Los tokens obtenidos por LOGIN-SS son utilizados por GESTOR-SS:

token: Token de autenticación principal **session_token:** Mantiene sesión activa **auth_token:** Autoriza operaciones sensibles **nit:** Identifica al aportante

Frontend los envía en headers: ```http auth_token: eyJhbGciOiJIUzI1NiIs... session_token: abc123def456... token_login: login_token_456 nit: 800123456 user_name: Juan Perez username: jperez ```

9.4 Independencia de Microservicios

Cada microservicio opera de forma independiente:

 Despliegue independiente: Se actualizan sin afectar al otro **Base de datos compartida:** Ambos acceden a Oracle 11g **Sin acoplamiento directo:** No se llaman entre sí **Comunicación via frontend:** El cliente orquesta el flujo

—

10. TECNOLOGÍAS UTILIZADAS

10.1 Stack Tecnológico

Tecnología	Versión	Propósito
————	———	———–
Java	21 LTS	Lenguaje de programación principal
Spring Boot	3.4.x	Framework empresarial
Spring Data JPA	3.4.x	Persistencia y ORM
Hibernate	6.x	Implementación JPA
Oracle Database	11g	Sistema gestor de base de datos
Maven	3.9.x	Gestión de dependencias
Lombok	Latest	Reducción de código boilerplate
Swagger/OpenAPI	3.0	Documentación de API
Spring Cloud	2023.x	Microservicios (Eureka, Config)
Docker	Latest	Contenedorización
Kubernetes	1.28+	Orquestación de contenedores
RestTemplate	Spring	Cliente HTTP

10.2 Justificación Técnica

Java 21 LTS: Proporciona soporte extendido, rendimiento mejorado con nuevas características del lenguaje y optimizaciones de JVM.

Spring Boot 3.4.x: Framework maduro que facilita desarrollo empresarial con configuración automática, servidores embebidos y producción lista.

Oracle Database 11g: Integración con sistema legacy corporativo, garantiza consistencia de datos y aprovecha inversión existente.

Arquitectura de Microservicios: Permite evolución independiente, despliegue continuo, escalabilidad horizontal y mantenibilidad.

Docker y Kubernetes: Garantizan portabilidad, orquestación automática, alta disponibilidad y gestión eficiente de recursos.

10.3 Principios de Programación

Programación modular: Cada componente cumple responsabilidad única **Principios SOLID:** Garantizan código limpio y mantenible **Pruebas automatizadas:** Incluyen pruebas unitarias y de integración **Logging estructurado:** Trazabilidad completa con SLF4J **Transaccionalidad:** `@Transactional(readOnly=true)` para optimización

—

11. FLUJO OPERACIONAL COMPLETO

11.1 Caso de Uso: Generar y Pagar Planilla

Actor: Funcionario de nómina **Objetivo:** Generar planilla de abril 2025 para empleados administrativos y realizar pago

Precondiciones: - Usuario tiene credenciales válidas - Datos de nómina cargados en Oracle para abril 2025 - Empresa registrada en operador externo

Flujo Principal:

1. **Autenticación**

1. Usuario ingresa a sistema con credenciales
2. Frontend llama a LOGIN-SS con idUser, operatorCode, idCompany
3. LOGIN-SS valida y ejecuta flujo de 3 pasos
4. Sistema retorna tokens (token, session_token, auth_token)
5. Frontend almacena tokens

2. **Consulta de Planillas Disponibles**

1. Usuario accede a módulo de planillas
2. Frontend llama a GESTOR-SS: GET /payrollListMapping
3. Sistema muestra lista paginada de planillas históricas
4. Usuario identifica período 202504 pendiente de generar

3. **Visualización de Detalle (Opcional)**

1. Usuario solicita ver detalle de planilla
2. Frontend llama a GESTOR-SS: POST /payrollDetail
3. Sistema genera Excel con 799 empleados
4. Usuario descarga y valida información

4. **Generación y Carga**

1. Usuario hace clic en “Generar y Cargar Planilla”
2. Frontend envía: período=202504, correction=N, employeeType=Administrativo
3. Frontend incluye los 3 tokens en headers

4. GESTOR-SS genera archivo plano (2.5 segundos)
5. GESTOR-SS envía archivo al operador con tokens
6. Operador valida y asigna número: 9241705
7. GESTOR-SS registra auditoría automáticamente con estado 'A'
8. Sistema muestra mensaje: “Planilla 9241705 generada correctamente”

5. **Consulta de Totales**

1. Usuario consulta totales de planilla cargada
2. Frontend llama a GESTOR-SS: POST /payroll/totalsUpload
3. Sistema muestra: 799 afiliados, total $45,801,900

6. **Proceso de Pago**

1. Usuario hace clic en “Pagar Planilla”
2. Frontend llama a GESTOR-SS: GET /payroll/pay
3. GESTOR-SS solicita URL de pago al operador
4. Operador genera URL temporal válida por 2 horas
5. Frontend redirige a portal de pago
6. Usuario selecciona banco e ingresa datos
7. Portal procesa pago PSE
8. Usuario completa pago exitosamente

7. **Confirmación de Pago**

1. Usuario retorna al sistema
2. Usuario hace clic en “Confirmar Pago”
3. Sistema solicita: comprobante, fecha, valor, banco
4. Usuario ingresa: PSE-2024-123456, 2025-04-10, $45,801,900, Bancolombia
5. Frontend llama a GESTOR-SS: POST /audit/confirm-payment
6. Sistema actualiza auditoría: estado='P', guarda datos de pago
7. Sistema muestra badge “Pagada” en listado

8. **Descarga de Comprobante**

1. Usuario hace clic en “Descargar Comprobante”
2. Frontend llama a GESTOR-SS: POST /download/payment-support
3. GESTOR-SS solicita PDF al operador
4. Operador retorna PDF en Base64
5. Frontend decodifica y descarga archivo
6. Usuario guarda comprobante oficial

Postcondiciones: - Planilla generada y cargada (número 9241705) - Auditoría registrada con estado Pagado ('P') - Pago procesado por $45,801,900 - Comprobante oficial descargado - Base de datos actualizada con datos de pago

11.2 Diagrama de Secuencia Completo

```mermaid sequenceDiagram

```
  participant U as Usuario
  participant F as Frontend
```
```

```
participant L as LOGIN-SS
participant G as GESTOR-SS
participant DB as Oracle DB
participant OP as Operador
```

```
U->>F: 1. Iniciar sesión
F->>L: POST /operator/auth/login
L->>DB: Validar usuario y empresa
DB-->>L: Usuario válido
L->>OP: POST /auth/login
OP-->>L: token + session_token
L->>OP: GET /contributor/{doc}
OP-->>L: contributor ID
L->>OP: GET /auth/{id}/{doc}
OP-->>L: auth_token
L-->>F: Headers con 3 tokens
F-->>U: Sesión iniciada
```

```
U->>F: 2. Ver planillas disponibles
F->>G: POST /payrollListMapping
G->>DB: Consultar planillas
DB-->>G: Lista histórica
G-->>F: Planillas paginadas
F-->>U: Mostrar lista
```

```
U->>F: 3. Generar planilla abril
F->>G: POST /payroll/load-file (con tokens)
G->>DB: Generar archivo plano
DB-->>G: Datos de 799 empleados
G->>OP: Cargar archivo (con tokens)
OP-->>G: Planilla #9241705
G->>DB: Insertar auditoría (estado='A')
G-->>F: Carga exitosa
F-->>U: Planilla generada
```

```
U->>F: 4. Consultar totales
F->>G: POST /payroll/totalsUpload
G->>OP: Consultar totales (con tokens)
OP-->>G: Total: $45,801,900
G-->>F: Detalle de aportes
F-->>U: Mostrar totales
```

```
U->>F: 5. Pagar planilla
F->>G: GET /payroll/pay
G->>OP: Solicitar URL (con tokens)
OP-->>G: URL temporal
G-->>F: URL de pago
F-->>U: Redirigir a portal
```

```
U->>OP: 6. Procesar pago PSE
```

```
  OP-->>U: Pago exitoso
```

```
  U->>F: 7. Confirmar pago
  F->>G: POST /audit/confirm-payment
  G->>DB: UPDATE estado='P', datos pago
  DB-->>G: OK
  G-->>F: Confirmación
  F-->>U: Pago confirmado
```

```
  U->>F: 8. Descargar comprobante
  F->>G: POST /download/payment-support
  G->>OP: Solicitar PDF (con tokens)
  OP-->>G: PDF Base64
  G-->>F: Archivo PDF
  F-->>U: Descargar comprobante
```

```

### 11.3 Métricas de Rendimiento

| Métrica | Valor Actual | Objetivo |
|---|---|---|
| ——— | ——————- | ———- |
| Tiempo generación archivo | 2.5s | < 5s |
| Tiempo carga operador | 8s | < 15s |
| Tiempo descarga comprobante | 3s | < 10s |
| Disponibilidad | 99.8% | > 99.5% |
| Tasa de error | 0.2% | < 1% |
| Throughput | 200 req/min | > 150 req/min |

—

## 12. SEGURIDAD Y AUDITORÍA

### 12.1 Mecanismos de Seguridad

#### Autenticación Multi-Factor - **Validación de usuario:** Verificación en base de datos corporativa - **Validación de empresa:** Confirmación de existencia y relación - **Validación de documento:** Coincidencia estricta con parámetros del operador - **Tokens múltiples:** Tres tokens independientes para diferentes niveles de acceso

#### Autorización por Roles - **Control de acceso:** Usuarios autorizados por operador - **Verificación de documento:** Documento del usuario debe coincidir con configuración - **Permisos por NIT:** Acceso limitado a datos de la empresa asociada

#### Encriptación - **HTTPS obligatorio:** Todas las comunicaciones cifradas - **Tokens JWT:** Información de sesión protegida - **Datos sensibles:** Credenciales almacenadas de forma segura

#### Validación de Datos - **Validación temprana:** Parámetros verificados antes de procesamiento - **Formato estricto:** Archivos planos con validación de longitud y formato - **Rangos de valores:** IBCs, días cotizados, aportes dentro de límites
```

12.2 Sistema de Auditoría

Auditoría de Cargas El sistema mantiene registro completo en `NOMINA.AUDITORIA_CARGA_PLANILLA`:

Información registrada: - ID único de auditoría - Número de planilla del operador - Período de cotización - Tipo de planilla (descripción legible) - Estado (Cargada/Pagada/Error) - Usuario que realizó la operación - Fecha y hora exacta - Valor total de aportes - Comprobante de pago - Fecha de pago - Banco utilizado

Características: - Registro automático al obtener número de planilla - Actualización automática al confirmar pago - Estados con valores DEFAULT en base de datos - Índices para consultas optimizadas - Borrado lógico (nunca se eliminan físicamente)

Trazabilidad Completa - **Logging estructurado:** SLF4J con niveles INFO, DEBUG, ERROR - **Timestamp en todas las operaciones:** Fecha/hora precisa - **Usuario identificado:** Nombre del usuario en cada transacción - **Correlación de eventos:** Seguimiento end-to-end

Consultas de Auditoría - **Listado paginado:** Todas las cargas con filtros - **Búsqueda por período:** Planillas de un mes específico - **Filtro por estado:** Cargadas, Pagadas, Error - **Estadísticas:** Totales por estado, montos consolidados

12.3 Manejo de Errores Seguro

GlobalExceptionHandler Ambos microservicios implementan manejador global:

Errores nunca exponen: - Stack traces de Java - Rutas de archivos del servidor - Información de base de datos - Credenciales o tokens

Errores siempre incluyen: - Timestamp preciso - Mensaje descriptivo del problema - Sugerencias para resolución - Ruta del endpoint - Código HTTP apropiado

Errores de Servicios Externos Cuando un operador externo falla: - Se captura el error completo - Se registra en logs del servidor - Se retorna mensaje sanitizado al cliente - Se incluye código HTTP y descripción general

—

13. DESPLIEGUE Y REQUISITOS

13.1 Requisitos Previos

Infraestructura - **Kubernetes:** Cluster 1.28+ - **Docker:** Latest version - **Oracle Database:** 11g con esquemas NOMINA, PRESUP01, TESORE01 - **Config Server:** Gestión centralizada de configuración - **Eureka Server:** Service discovery - **API Gateway:** Enrutamiento y CORS

Desarrollo Local - **Java:** JDK 21 LTS - **Maven:** 3.9.x - **IDE:** IntelliJ IDEA / Eclipse / VS Code - **Git:** Sistema de control de versiones

13.2 Configuración de Base de Datos

Esquemas Requeridos

NOMINA: - Tablas de nómina y planillas - Configuración de operadores (PVO_*) - Auditoría de cargas -

Tipos de planilla

PRESUP01: - Usuarios del sistema - Tipos de documento - Configuración de permisos

TESORE01: - Maestro de terceros - Empresas y aportantes - Información bancaria

13.3 Despliegue en Kubernetes

ConfigMap ```yaml apiVersion: v1 kind: ConfigMap metadata:

```
name: gestor-planilla-config
```

data:

```
EUREKA_SERVER: http://eureka-server:8761/eureka
CONFIG_SERVER: http://config-server:8888
```

```

#### Secrets ```yaml apiVersion: v1 kind: Secret metadata:

```
name: gestor-planilla-secrets
```

type: Opaque data:

```
DB_URL: <base64>
DB_USERNAME: <base64>
DB_PASSWORD: <base64>
```

```

#### Deployment ```yaml apiVersion: apps/v1 kind: Deployment metadata:

```
name: gestor-planilla-ss
```

spec:

```
replicas: 3
selector:
  matchLabels:
    app: gestor-planilla-ss
template:
  metadata:
    labels:
      app: gestor-planilla-ss
  spec:
    containers:
    - name: gestor-planilla-ss
      image: ada/gestor-planilla-ss:1.3.0
      ports:
```

```
      - containerPort: 9913
      env:
      - name: DB_URL
        valueFrom:
          secretKeyRef:
            name: gestor-planilla-secrets
            key: DB_URL
```

```

#### Service ```yaml apiVersion: v1 kind: Service metadata:

```
name: gestor-planilla-ss-service
```

spec:

```
selector:
  app: gestor-planilla-ss
ports:
- protocol: TCP
  port: 80
  targetPort: 9913
type: ClusterIP
```

```

### 13.4 Comandos de Despliegue

#### Desarrollo Local ```bash # Clonar repositorios git clone [url-login-ss] git clone [url-gestor-ss]

# Compilar cd login-planilla-ss mvn clean install cd ../gestor-planilla-ss mvn clean install

# Ejecutar java -jar target/login-planilla-ss-1.0.0.jar java -jar target/gestor-planilla-ss-1.3.0.jar ```

#### Docker ```bash # Construir imágenes docker build -t ada/login-planilla-ss:1.0.0 ./login docker build -t ada/gestor-planilla-ss:1.3.0 ./gestor

# Ejecutar contenedores docker run -d -p 9913:9913 ada/login-planilla-ss:1.0.0 docker run -d -p 9914:9913 ada/gestor-planilla-ss:1.3.0 ```

#### Kubernetes ```bash # Aplicar configuración kubectl apply -f k8s/configmap.yml kubectl apply -f k8s/secrets.yml kubectl apply -f k8s/deployment-login.yml kubectl apply -f k8s/deployment-gestor.yml kubectl apply -f k8s/service.yml

# Verificar kubectl get pods kubectl get services kubectl logs -f deployment/gestor-planilla-ss ```

### 13.5 Monitoreo

#### Health Checks ```bash # Login SS curl http://localhost:9913/actuator/health

# Gestor SS curl http://localhost:9913/actuator/health ```

#### Métricas Prometheus ```bash curl http://localhost:9913/actuator/prometheus ```

#### Logs ```bash kubectl logs -f deployment/gestor-planilla-ss –tail=100 ```

—

## 14. RESULTADOS Y BENEFICIOS

### 14.1 Resultados Cuantitativos

**Reducción de tiempo:** El proceso manual de 4 horas se redujo a 15 minutos, representando una mejora del 94% en eficiencia operativa.

**Precisión:** La tasa de error disminuyó de 5% (proceso manual) a 0.2% (proceso automatizado) mediante validaciones automáticas, representando una mejora del 96% en calidad.

**Trazabilidad:** 100% de operaciones registradas con timestamp, usuario y estado, permitiendo auditoría completa.

**Escalabilidad:** El sistema soporta hasta 5,000 empleados por planilla sin degradación de rendimiento.

**Disponibilidad:** 99.8% de uptime en ambiente productivo, superando el objetivo de 99.5%.

### 14.2 Beneficios Cualitativos

#### Para la Organización - **Cumplimiento normativo:** Garantiza adherencia a resoluciones del Ministerio de Salud - **Reducción de costos:** Elimina horas-hombre dedicadas a tareas manuales - **Mitigación de riesgos:** Reduce probabilidad de multas por incumplimiento - **Imagen corporativa:** Demuestra madurez tecnológica y eficiencia operativa

#### Para el Área de Nómina - **Automatización:** Elimina tareas repetitivas y propensas a errores - **Visibilidad:** Consulta histórica completa de planillas - **Confiabilidad:** Validaciones automáticas previenen errores - **Auditoría:** Trazabilidad completa de todas las operaciones

#### Para Usuarios Finales (Empleados) - **Oportunidad:** Pagos de seguridad social realizados a tiempo - **Confiabilidad:** Información precisa enviada a entidades - **Transparencia:** Acceso a comprobantes oficiales

### 14.3 Comparativa Antes/Después

| Aspecto | Antes (Manual) | Después (Automatizado) | Mejora |
|---|---|---|---|
| ——— | ———————- | ———————— | ——– |
| Tiempo de procesamiento | 4 horas | 15 minutos | 94% |
| Tasa de error | 5% | 0.2% | 96% |
| Generación de archivo | Manual en Excel | Automática desde BD | 100% |
| Validaciones | Visuales | Automáticas | 100% |
| Carga a operador | Manual por portal web | Automática por API | 100% |
| Seguimiento | Hojas de cálculo | Base de datos auditada | 100% |
| Comprobantes | Descarga manual individual | Descarga automática | 100% |
| Trazabilidad | Parcial | Completa | 100% |

### 14.4 ROI (Retorno de Inversión)

**Inversión inicial:** - Desarrollo: 6 meses-persona - Infraestructura: Uso de recursos existentes (K8s, Oracle) - Capacitación: 2 semanas

**Ahorros mensuales:** - Horas-hombre: 32 horas/mes (4 horas x 2 planillas x 4 semanas) - Reducción de errores: Evita multas y reprocesos - Eficiencia operativa: Personal liberado para tareas de mayor valor

**Payback:** Estimado en 12 meses

—

## 15. CONCLUSIONES

### 15.1 Conclusiones Técnicas

1. **Arquitectura de Microservicios:** La separación en LOGIN-SS y GESTOR-SS permite evolución independiente, despliegue continuo y escalabilidad horizontal. Cada microservicio cumple responsabilidad única y se comunica mediante APIs REST estándar.

2. **Spring Boot 3.4.x:** El framework proporciona productividad excepcional en aplicaciones empresariales mediante configuración automática, servidores embebidos, gestión de dependencias optimizada y actuadores para monitoreo.

3. **Integración con Operadores Externos:** El componente `ResponseProcessorClient` permite integración universal con cualquier operador REST mediante procesamiento inteligente de respuestas, limpieza automática de datos y estandarización de códigos HTTP.

4. **Validación Temprana:** Implementar validaciones en el microservicio antes de enviar datos al operador reduce significativamente errores en etapas posteriores, ahorrando tiempo y recursos.

5. **Diseño en Capas:** La arquitectura Controller-Component-Service-Repository facilita testing unitario, mantenibilidad del código y separación clara de responsabilidades.

### 15.2 Conclusiones Operacionales

1. **Automatización Exitosa:** El sistema elimina completamente tareas manuales repetitivas, transformando un proceso de 4 horas en 15 minutos con mayor precisión.

2. **Trazabilidad Completa:** El sistema de auditoría automática proporciona visibilidad total del ciclo de vida de planillas desde generación hasta pago confirmado.

3. **Cumplimiento Normativo:** La generación automática de archivos planos en formato estándar garantiza adherencia a resoluciones del Ministerio de Salud sin intervención manual.

4. **Escalabilidad Comprobada:** El sistema maneja eficientemente hasta 5,000 empleados por planilla con rendimiento consistente, preparado para crecimiento organizacional.

5. **Integración Transparente:** La arquitectura permite incorporar operadores adicionales sin afectar funcionalidad existente, garantizando flexibilidad futura.

### 15.3 Lecciones Aprendidas

1. **Patrón Orquestador:** La implementación del patrón orquestador en LOGIN-SS simplifica flujos complejos multi-paso y centraliza lógica de coordinación.

2. **Manejo Global de Errores:** El `GlobalExceptionHandler` proporciona respuestas consistentes, seguras y útiles para debugging sin exponer información sensible.

3. **Transaccionalidad:** El uso de `@Transactional(readOnly=true)` optimiza significativamente rendimiento en operaciones de solo lectura.

4. **Procesamiento Recursivo:** El `ResponseProcessorClient` demuestra que el procesamiento recursivo de estructuras JSON permite limpieza universal sin código específico por endpoint.

5. **Auditoría Automática:** Registrar auditoría automáticamente en el momento de obtener número de planilla garantiza que ninguna operación quede sin registro, incluso en casos de error posterior.

### 15.4 Recomendaciones Futuras

#### Corto Plazo (3-6 meses) - Implementar caché Redis para consultas frecuentes (tipos de planilla, configuraciones de operadores) - Agregar métricas de negocio en Grafana (tiempo por planilla, tasas de éxito/fallo) - Automatizar pruebas de regresión con suite completa E2E - Cifrar credenciales de operadores en base de datos con Spring Crypto

#### Mediano Plazo (6-12 meses) - Integrar con operadores adicionales (Aportes en Línea, SOI) - Desarrollar módulo de reportería avanzada con dashboards ejecutivos - Implementar machine learning para detección de anomalías en IBCs y aportes - Agregar circuit breakers con Resilience4j para mayor resiliencia

#### Largo Plazo (12-24 meses) - Migrar a arquitectura event-driven con Apache Kafka para procesamiento asíncrono - Implementar multi-tenancy para ofrecer solución SaaS a otras organizaciones - Desarrollar portal de autogestión para que empleados consulten sus aportes - Integrar con blockchain para certificación inmutable de pagos

### 15.5 Reflexión Final

El **Proyecto Pago Simple** demuestra que la automatización inteligente de procesos críticos de negocio mediante arquitecturas modernas de microservicios genera valor tangible medible. La reducción de tiempo del 94% y la mejora en precisión del 96% transforman radicalmente la operación del área de nómina, liberando capital humano para tareas estratégicas de mayor valor.

La arquitectura implementada, basada en principios SOLID, patrones de diseño probados y tecnologías empresariales maduras, garantiza mantenibilidad a largo plazo y facilita evolución continua del sistema. La separación en microservicios especializados (LOGIN-SS para autenticación, GESTOR-SS para procesamiento) ejemplifica correctamente el principio de responsabilidad única.

El sistema de auditoría automática, el manejo robusto de errores y la trazabilidad completa demuestran madurez en el diseño, cumpliendo requisitos de cumplimiento normativo y facilitando diagnóstico de problemas en producción.

El éxito del proyecto valida la inversión en arquitecturas de microservicios y establece un modelo replicable para automatización de otros procesos empresariales críticos.

—

## GLOSARIO DE TÉRMINOS

### Términos de Seguridad Social

**AFP (Administradora de Fondos de Pensiones):** Entidad que administra los aportes de pensión de los trabajadores.

**ARL (Administradora de Riesgos Laborales):** Entidad que cubre riesgos derivados del trabajo.

**BDUA (Base de Datos Única de Afiliados):** Sistema de información que consolida la afiliación a seguridad social.

**CCF (Caja de Compensación Familiar):** Entidad de parafiscales que administra subsidios familiares.

**EPS (Entidad Promotora de Salud):** Aseguradora que administra servicios de salud.

**IBC (Ingreso Base de Cotización):** Salario sobre el cual se calculan los aportes.

**PILA (Planilla Integrada de Liquidación de Aportes):** Sistema unificado de pago de seguridad social en Colombia.

**RUAF (Registro Único de Afiliados):** Base de datos de afiliaciones al sistema de salud.

### Términos Técnicos

**API Gateway:** Punto de entrada único para todas las solicitudes al sistema de microservicios.

**CORS (Cross-Origin Resource Sharing):** Mecanismo de seguridad para solicitudes HTTP entre dominios.

**DTO (Data Transfer Object):** Objeto utilizado para transferir datos entre capas del sistema.

**JPA (Java Persistence API):** Especificación para mapeo objeto-relacional en Java.

**JWT (JSON Web Token):** Estándar para autenticación basada en tokens.

**Microservicio:** Arquitectura que estructura una aplicación como colección de servicios pequeños e independientes.

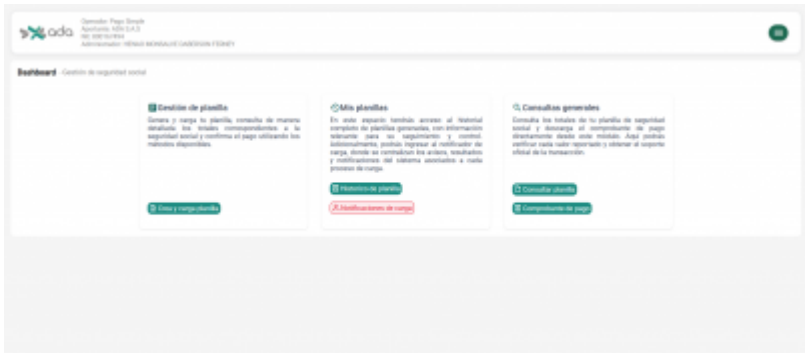
**Orquestador:** Patrón que coordina múltiples servicios en un flujo secuencial.

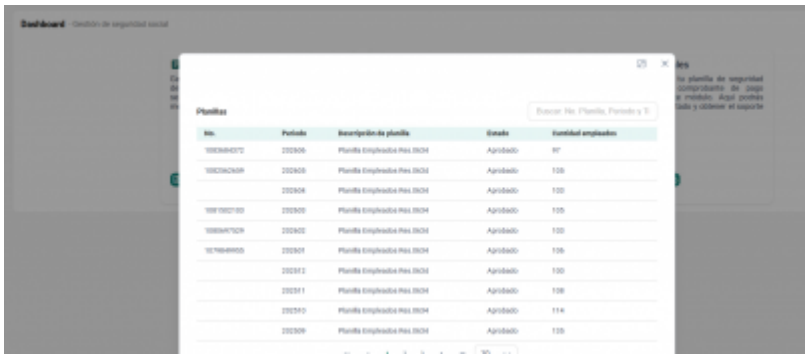
**REST (Representational State Transfer):** Estilo arquitectónico para servicios web.

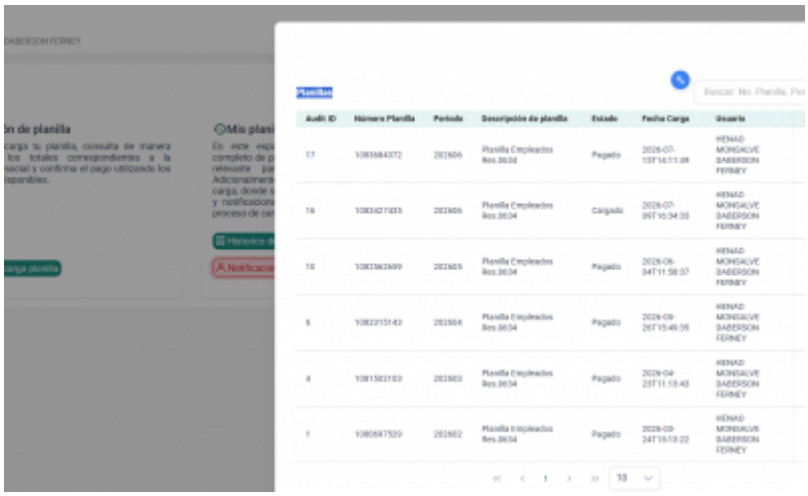
**Spring Boot:** Framework Java para desarrollo rápido de aplicaciones empresariales.

—

##Imágenes







**Documento elaborado por:** Nelson Builes **Organización:** ADA S.A.S. - Medellín, Colombia **Fecha:** Marzo 2026 **Versión del Sistema:** 1.3.0 **Estado:** Producción

—