

Anexo A - Sistema de Gestión de Trámites

0 INDICE

0	INDICE	1
1	Introducción.....	5
1.1	Propósito del documento	5
1.2	Alcance.....	5
1.2.1	Posicionamiento del producto	5
1.3	Puntos a ser analizados en mayor profundidad.....	5
1.4	Partes interesadas.....	6
1.5	Supuestos principales	6
1.5.1	Procesos actualizados conforme al nuevo Sistema de Gestión de la Calidad (SGC)	6
1.6	Documentación de referencia	7
1.6.1	Procedimientos internos pertenecientes al SGC.....	7
1.6.2	Documentos de referencia externa.....	7
1.7	Objetivos de calidad de la arquitectura	8
2	Requisitos	10
2.1	Requisitos funcionales	10
2.2	Requisitos no funcionales.....	11
2.2.1	Metodología de trabajo y flujo DevOps (Repositorio GitLab)	12
3	Contexto.....	14
3.1	Visión general y objetivos.....	14
3.1.1	Actores principales.....	15
3.2	Objetivos clave	15
4	Estrategia de solución	15
4.1	Decisiones tecnológicas	15
4.2	Descomposición del sistema.....	15
4.3	Logro de objetivos de calidad	16
4.4	Decisiones organizacionales.....	16

5	Componentes.....	16
5.1	Flujo de Interacción General.....	17
5.2	Componentes principales y funciones.....	18
5.3	Flujo de un trámite (ejemplo).....	19
6	Modulos	20
6.1	Frontend.....	20
6.2	Backend.....	20
6.3	Motor de tramites	21
6.3.1	Requisitos específicos del motor de flujo:	21
6.4	Autenticación y autorización	22
6.4.1	Autenticación y gestión de identidades (flujo diferenciado):.....	22
6.4.2	Secuencia resumida de interacción por tipo de usuario:	23
6.4.3	Consideraciones de seguridad aplicadas al flujo:	23
6.5	Sistema de firma digital remota	23
6.6	Cumplimiento de requisitos funcionales.....	24
7	Arquitectura ampliada – Motor de tramites	26
7.1	Propósito y alcance	26
7.2	Composición interna del módulo	26
7.3	Modelo de flujo: grafo de nodos conectados.....	27
7.4	Tipos de nodos y su interacción	27
7.5	Asignación de responsabilidades y responsable máximo.....	28
7.6	Estados genéricos del trámite (vista externa e interna).....	28
7.7	Excepciones y exceptuación de nodos	29
7.7.1	Modelo de exceptuación por nodo Cada nodo define:	29
7.7.2	Transacción de excepción auditable La acción de exceptuar debe:.....	29
7.8	Auditoría, firma y reconstrucción completa del historial	29
7.8.1	Recomendación de arquitectura de auditoría	29
7.8.2	Firma.....	30
7.9	Versionado de plantillas y compatibilidad hacia atrás	30
7.10	Persistencia de datos dinámicos (Plantilla e Instancia).....	30
7.11	Interfaz de modelado y alternativa de implementación (BPM externo vs motor propio)	30
7.12	Flujos operativos clave.....	31

7.12.1	Inicio de un trámite.....	31
7.12.2	Ejecución de nodos automáticos (condición/procesamiento)	31
7.12.3	Ejecución de nodos manuales	31
7.12.4	Excepción	31
7.12.5	Clonado (para empresas)	32
7.13	Requisitos no funcionales específicos del motor	32
7.14	Ejemplo de plantilla para tramite de inicio	32
8	Entidades y Flujos principales.....	33
8.1	Diagrama entidad relacion inicial propuesto.....	34
8.2	Modelo de datos modulo motor de tramites	36
8.2.1	Modelo de Datos: objetivos, alcance y principios.....	36
8.2.2	Stack de persistencia y criterios generales	36
9.1.2	Versionado explícito de TipoTramite (plantilla de flujo).....	37
9.1.3	Estructuras dinámicas: formularios, reglas y datos de instancia	37
9.1.4	Estrategia documental: unificación vs tablas tipificadas	38
9.1.5	Índices principales (recomendación inicial).....	38
9.1.6	Consultas operativas y analíticas prioritarias.....	39
9.1.7	Integridad, consistencia y concurrencia.....	39
9.1.8	Retención, archivado y borrado lógico.....	40
9.1.9	Recomendaciones específicas para búsqueda	40
9.1.10	Diccionario de datos (entidades principales)	40
9.1.11	Patrones de consulta (consultas principales)	41
9.1.12	Consideraciones de performance y escalabilidad	43
9.1.13	Relaciones polimórficas: asociación de Trámite con entidades variables	43
10	Despliegue	45
10.1	Infraestructura de despliegue con Docker Compose (On-Premise).....	46
11	Decisiones de arquitectura	50
11.1	Repositorio y control de versiones (GitLab)	50
11.2	Justificación de las decisiones técnicas.....	51
11.2.1	Arquitectura modular basada en contenedores (Docker) con orquestación mediante Docker Compose	51
11.2.2	Aplicación principal como monolito	52
11.2.3	Servicios separados para funciones transversales (auditoría, logs, mantenimiento).....	52

11.2.4	Uso de tecnologías modernas y open source	52
11.2.5	Autenticación y Autorización centralizada con OAuth2 y OIDC	53
11.2.6	Modelo de autorización basado en roles y habilidades (RBAC + ABAC)	53
11.2.7	Observabilidad y logs	54
11.3	Limitaciones	54
12	Glosario.....	55
12.1	Glosario modulo motor de tramites:.....	58

1 INTRODUCCIÓN

1.1 PROPÓSITO DEL DOCUMENTO

Este documento establece la arquitectura y los requisitos del Sistema Integral de Gestión de Trámites (SIGT) para la Secretaría de Industria, Promoción Económica y Calidad Territorial de Tierra del Fuego, Antártida e Islas del Atlántico Sur (en adelante, SiyPE TDF). Su objetivo es describir la estructura técnica, los actores involucrados, los procesos soportados y los criterios de calidad que guiarán el diseño y desarrollo del sistema, de acuerdo con el estándar arc42.

1.2 ALCANCE

SIGT automatizará el ciclo de vida completo de los trámites gestionados por la SiyPE TDF: ingreso, revisión, aprobación, seguimiento y emisión de certificados (incluido el Certificado de Origen, CEOR). El sistema incorporará las nuevas definiciones de proceso resultantes de la certificación ISO 9001, sin contemplar la migración de datos de sistemas actuales.

1.2.1 Posicionamiento del producto

Elemento	Declaración
Para	Funcionarios, personal técnico, usuarios administrativos de la Secretaría de Industria de Tierra del Fuego y representantes de las empresas.
Quien	Obtendrán un sistema integrado, confiable y alineado a los procesos certificados de calidad, que mejora la trazabilidad, eficiencia y control de los trámites.
SGIT	Sistema informático integral para la gestión de trámites administrativos y de inspección.
Que	Permite gestionar el ciclo completo de los trámites, integrarse con sistemas externos clave (AREF, ARCA, SGC), y generar indicadores y reportes operativos.
Sin ser	Las soluciones actuales basadas en procedimientos manuales, hojas de cálculo dispersas o sistemas parciales sin integración ni trazabilidad estándar.
El sistema	Aporta una arquitectura modular, extensible y segura, con trazabilidad completa, integración a los sistemas del ecosistema institucional y alineamiento con ISO 9001.

Tabla 1: Posicionamiento de producto del sistema de gestión de tramites

1.3 PUNTOS A SER ANALIZADOS EN MAYOR PROFUNDIDAD

Las integraciones con sistemas externos tales como ARCA (Aduana y ex-AFIP), AREF y otros sistemas orientados específicamente a extracción de datos – reportes y KPIs serán objeto de análisis en etapas posteriores del diseño. Su incorporación efectiva dependerá de dos factores clave:

- la capacidad técnica actual de dichos sistemas para permitir una integración segura y eficiente,

- y la necesidad concreta de dicha integración, la cual se definirá con mayor precisión a partir del análisis detallado de los casos de uso específicos.

Este enfoque busca asegurar que las decisiones de integración respondan tanto a criterios de viabilidad técnica como de valor funcional dentro del contexto operativo del sistema a desarrollar.

1.4 PARTES INTERESADAS

Stakeholder	Rol	Intereses principales
SlyPE TDF	Propietario del sistema	Agilidad, trazabilidad, cumplimiento normativo y gestión
Empresas subrégimen	Usuarios principales	Usabilidad, rapidez, transparencia
Empresas CEOR	Usuarios principales	Acceso claro a trámites de exportación
Agencia de Innovación (Infra IT)	Proveedor de infraestructura y soporte inicial	Despliegue eficiente - simple, escalabilidad
Dependencias gubernamentales	Participantes en flujos de CEOR	Integración transparente, seguridad y usabilidad
AREF, ARCA	Revisión documental	Disponibilidad de datos actualizados, integridad

Tabla 2: Actores principales

1.5 SUPUESTOS PRINCIPALES

- El sistema debe desplegarse on-premise o en nube según disponibilidad de la Agencia de Innovación.
- No se migrarán datos históricos, únicamente se parametriza la estructura actual.
- Los procesos – tramites – a mapear serán los actualizados para el nuevos SGC.
- Los requerimientos de los tramites mapeados tienen que poder ser modificados sin cambios significativos en la aplicación. Idealmente serán configurados por el propietario del sistema.

1.5.1 Procesos actualizados conforme al nuevo Sistema de Gestión de la Calidad (SGC)

En paralelo a la elaboración de este documento, se encuentra en curso el proceso de certificación de calidad de los procedimientos de la Secretaría de Industria, bajo el estándar ISO 9000. Este trabajo implica la revisión, actualización y mejora integral de los procesos administrativos y técnicos que intervienen en la gestión de trámites de la dependencia.

Como resultado, se asume que los procesos a mapear e informatizar en el marco del presente sistema serán aquellos revisados y validados como parte del nuevo Sistema de Gestión de la Calidad (SGC). Si bien este documento no forma parte del SGC en términos formales, se desarrollará en coherencia con sus buenas prácticas y tomará como referencia la documentación oficial generada en dicho marco, referenciándose siempre que sea utilizada.

Por lo tanto, el diseño del nuevo sistema informático deberá alinearse con los procedimientos actualizados y ser capaz de integrarse de forma armónica con el sistema de calidad certificado, constituyéndose así en un soporte operativo clave para su implementación efectiva y sostenida.

1.6 DOCUMENTACIÓN DE REFERENCIA

El núcleo de requisitos funcionales específicos que delinean el sistema está determinado en los procedimientos de los tramites a ser mapeados.

1.6.1 Procedimientos internos pertenecientes al SGC

Tramite	Procedimiento
Inicio	SITDF-P-20-07
Reinicio	SITDF-P-21-07
Ampliaciones	SITDF-P-22-05
Desafectaciones	SITDF-P-23-05, SITDF-I-23-01-01
Repuestos	SITDF-P-24-08
Scrap	SITDF-P-25-04
Fazon	SITDF-P-26-04
Control de cumplimiento de proyectos	SITDF-P-29-05, SITDF-I-29-01-01
Mesa de Entradas	SITDF-P-30-07, SITDF-I-30-01-02, SITDF-I-30-02-02, SITDF-I-30-03-02
Control de cumplimiento de la TVPP	SITDF-P-28-03, SITDF-I-20-04
Acreditación de origen	SITDF-P-31-00, SITDF-P-32-00
Certificado de origen	SITDF-P-34-00

Tabla 3: Principales procedimientos del SGC

1.6.2 Documentos de referencia externa

Estándar	Descripción breve	Implicaciones en la arquitectura
arc42 + C4 Model	Plantilla y enfoque estructurado para documentar arquitecturas de software, integrando vistas lógicas, técnicas y de proceso. El modelo C4 permite representar distintos niveles de abstracción (Contexto, Contenedores, Componentes, Código).	Se utilizará la estructura de arc42 como base para documentar la arquitectura, incorporando diagramas C4 para representar de forma clara y jerárquica la composición del sistema.
ISO 9001	Estándar internacional para Sistemas de Gestión de la Calidad (SGC). Define requisitos para asegurar la consistencia y mejora de procesos y la satisfacción del cliente.	La arquitectura debe facilitar la trazabilidad de los procesos, la generación de informes de calidad y la integración con el SGC certificado, garantizando que todos los flujos cumplan con controles de calidad.
ISO/IEC 42010	Define los principios para la descripción de arquitecturas de sistemas de software, incluyendo vistas, modelos y correspondencia entre ellas.	Se estructurará la documentación arquitectónica en vistas (contexto, lógica, procesos, implementación, etc.) y se mantendrán relaciones claras entre artefactos.
ISO/IEC 25010	Especifica el modelo de calidad de producto de software, abarcando características como funcionalidad, fiabilidad, usabilidad, rendimiento y seguridad.	Los requisitos de calidad se traducirán en escenarios medibles y métricas concretas, como se recoge en la sección de Quality goals, y afectarán decisiones de

		diseño (caching, redundancia, autenticación).
ISO/IEC 27001	Estándar para Sistemas de Gestión de Seguridad de la Información (SGSI). Establece requisitos para proteger la confidencialidad, integridad y disponibilidad.	La arquitectura incluirá controles de seguridad (autenticación, autorización, cifrado en tránsito y en reposo) y procesos de auditoría, así como la segregación de datos según clasificación.
OWASP Application Security Verification Standard (ASVS) Nivel 2	Marco de referencia para evaluar la seguridad de aplicaciones web, agrupando requisitos de desarrollo seguro y pruebas de vulnerabilidades.	Los componentes de la capa de presentación y negocio deberán cumplir con validación de entradas, gestión de sesiones seguras y protección contra vulnerabilidades críticas (XSS, SQLi, CSRF).
REST/JSON API Best Practices	Conjunto de recomendaciones para diseñar APIs RESTful, enfatizando consistencia en URIs, uso de verbos HTTP, manejo de errores y versionado.	Se definirá un contrato claro de API, con rutas coherentes, códigos de estado adecuados, documentación automática (OpenAPI/Swagger) y mecanismos de versionado para evolutividad del servicio.
W3C WCAG 2.1	Directrices de accesibilidad para contenidos web, enfocadas en tres niveles (A, AA, AAA) para garantizar que la interfaz sea accesible a usuarios con discapacidades.	El diseño de la interfaz deberá cumplir al menos nivel AA, implicando semántica HTML correcta, contraste de colores, navegación por teclado y etiquetas ARIA donde corresponda.

Tabla 4: Principales estándares y referencias externas

1.7 OBJETIVOS DE CALIDAD DE LA ARQUITECTURA

En esta sección se definen los objetivos de calidad de la arquitectura cuyo cumplimiento es de máxima importancia para los principales stakeholders. Cada objetivo se ilustra con un escenario concreto y un umbral de medición, ordenados por prioridad.

Prioridad	Objetivo de calidad	Escenario concreto	Métrica / Umbral	Motivación
1	Integridad funcional	El sistema debe completar el 100 % de los pasos definidos para los diferentes tramites sin errores ni excepciones no manejadas.	Cobertura funcional: 100 % en pruebas de aceptación de casos de uso.	Garantizar que todos los requisitos (funcionales y no funcionales) estén efectivamente implementados.
2	Fiabilidad y recuperación	Tras una indisponibilidad total (por fallo de infraestructura), el sistema debe	RTO ≤ 8 hr; RPO = 0 (ninguna transacción confirmada se pierde).	Minimizar el impacto operativo y asegurar continuidad del

		volver a estar operativo en menos de 8 hr y sin pérdida de transacciones confirmadas.		servicio frente a desastres.
3	Seguridad e integridad de datos	Todas las operaciones que leen o modifican datos sensibles (p. ej. exportación de informes KPI) deben pasar autenticación, autorización y validación de entrada según OWASP ASVS nivel 2.	100 % de endpoints sin vulnerabilidades críticas en escaneo ASVS nivel 2.	Proteger la confidencialidad e integridad de la información y cumplir normativa de seguridad.
4	Mantenibilidad y transferencia	Un equipo externo, tras 5 días de inducción, debe ser capaz de desplegar, diagnosticar y modificar el sistema usando la documentación asociada.	Tiempo de inducción ≤ 5 días	Facilitar la transferencia al gobierno y el mantenimiento a largo plazo por terceros.
5	Rendimiento y escalabilidad	Bajo una carga de 50 usuarios concurrentes realizando búsquedas y reportes, el 95 % de las respuestas a operaciones críticas debe completarse en menos de 2 s.	≥ 95 % de respuestas < 2 s con 50 usuarios simultáneos.	Asegurar una experiencia de usuario ágil y soportar el crecimiento de la carga de trabajo.
6	Observabilidad y trazabilidad end-to-end	Toda operación crítica (alta/modificación de trámite, firma, notificación,	Cobertura de logs JSON = 100 % en servicios/apps; 100 % de operaciones críticas incluyen correlationId; ≥ 95 % de spans de traza	Asegurar auditoría, diagnósticos rápidos y trazabilidad técnica/funcional

		integración) genera eventos de log estructurados (JSON) con correlationId y trazas exportadas hacia la solución interna de la Agencia y, en paralelo, hacia el stack ELK del proyecto para análisis y métricas.	exportados; latencia de ingesta de logs ≤ 5 min; retención según política de la Agencia (mín. 30 días), con preservación equivalente en ELK del proyecto.	, integrando con la solución institucional sin perder capacidad de análisis propia.
7	Firma digital remota (rendimiento y límites operativos)	El sistema firma/valida PDFs vía API Nación. Cuando un documento supera los límites de tamaño (MB) o páginas del servicio nacional, el flujo segmenta automáticamente y aplica optimización de PDF para cumplir los toques. O implementa firma mediante métodos alternativos.	Para documentos dentro de los límites publicados por el servicio nacional (L_MB, L_PAG): tiempo de firma/validación ≤ 10 s por documento; tasa de error $< 1\%$ en llamadas a la API. Para documentos que excedan L_MB o L_PAG: particionado/optimización automática con tiempo adicional ≤ 3 s por segmento y reintento con backoff (máx. 2 reintentos).	Garantizar una experiencia predecible y compatible con limitaciones técnicas del proveedor nacional, mitigando fallos por tamaño o paginado mediante segmentación y optimización.

Tabla 5: Objetivos clave de la calidad de arquitectura

2 REQUISITOS

2.1 REQUISITOS FUNCIONALES

Se cumplirá con todos los requisitos funcionales establecidos en los procedimientos internos indicados en la sección 1.6.1 “Procedimientos internos pertenecientes al SGC”, los cuales definen el alcance, funciones, actores, responsabilidades y flujo de todos los procesos que serán mapeados e incorporados al sistema informático. Estos requisitos funcionales se identificarán y documentarán a lo largo del desarrollo de la asistencia técnica y se relevan en

los documentos específicos de cada trámite mediante la elaboración de casos de uso detallados.

Los requisitos funcionales generales, o transversales a todo el sistema, se detallarán a lo largo de este documento, alcanzando mayor nivel de precisión y descripción a medida que avance sobre el mismo.

Asimismo, la plataforma deberá adecuarse a la normativa vigente de sistemas y procesos del gobierno provincial.

2.2 REQUISITOS NO FUNCIONALES

Los requisitos no funcionales que guían el diseño de la arquitectura del sistema informático se definen inicialmente a partir de criterios generales de calidad, seguridad y eficiencia, y serán complementados y ajustados durante el desarrollo de la asistencia técnica. Su validación y especificación detallada se realizará en conjunto con las áreas técnicas y funcionales intervinientes. Cada requisito no funcional contará con su propio apartado específico donde se describirán las condiciones de cumplimiento, métricas asociadas y su impacto en el diseño arquitectónico.

Los requisitos iniciales considerados son:

- **Confiabilidad:** Tolerancia de fallos críticos menor al 1%. Capacidad de recuperación ante desastres en un plazo máximo de 8 horas.
- **Seguridad:** Autenticación mediante SSO y protocolos estándar (OpenID Connect). Cifrado de datos en reposo y en tránsito.
- **Escalabilidad:** Soporte para al menos 50 transacciones concurrentes sin degradación significativa del rendimiento.
- **Mantenibilidad:** Documentación técnica y funcional exhaustiva. Plan de transferencia y capacitación para continuidad operativa.
- **Portabilidad:** Capacidad de despliegue mediante contenedores aislados (Docker – Podman), facilitando la replicabilidad del entorno.
- **Rendimiento:** Tiempo de respuesta menor a 1 segundo en las operaciones principales del sistema.
- **Retención de datos:** Almacenamiento de logs y registros de auditoría por un período mínimo de 5 años.
- **Seguridad de acceso:** Validación de entradas y controles de acceso basados en roles definidos.
- **Integridad de datos:** Realización de copias de seguridad diarias y mecanismos de verificación de integridad.
- **Disponibilidad:** Uptime garantizado del 99.9 % con mecanismos de monitoreo y alerta proactivos.
- **Compatibilidad de plataformas:** Aplicación web accesible desde los navegadores modernos más utilizados.
- **Interoperabilidad:** APIs RESTful con intercambio de datos en formato JSON para integración con sistemas externos.
- **Usabilidad:** Interfaz intuitiva y accesible, cumpliendo con WCAG 2.1 nivel AA.

- **Desplegabilidad:** Automatización de despliegue mediante scripts y configuraciones reproducibles en entornos encapsulados.
- **Hardware:** El sistema tendrá que poder ser ejecutado en una instancia de AWS m5.xlarge o equivalente en hardware on premise.
- **Observabilidad y auditoría:** Emisión de logs estructurados (JSON) en todas las operaciones críticas
- **Trazabilidad probatoria de firma:** Registro y conservación del ID de transacción del proveedor nacional, hash del documento previo y posterior a la firma, y (cuando corresponda) sello de tiempo y evidencias de OCSP/CRL para validación posterior.

2.2.1 Metodología de trabajo y flujo DevOps (Repositorio GitLab)

La plataforma y el proceso de desarrollo deberán ajustarse a una metodología de trabajo estandarizada sobre el GitLab de la Agencia, asegurando trazabilidad de cambios, control de calidad continuo y despliegues reproducibles. Las especificaciones finales de estructura de repositorio, permisos y convenciones serán provistas por la Agencia al inicio del desarrollo; hasta entonces, se aplican los siguientes requisitos mínimos obligatorios.

- Repositorio y permisos (GitLab de la Agencia): uso de proyectos y grupos definidos por la Agencia; ramas protegidas (main, release/*); Code Owners para áreas críticas; acceso bajo RBAC.
- Flujo de ramas y versionado: esquema GitFlow o Trunk-Based (a confirmar con la Agencia); ramas feature/*, bugfix/*, hotfix/*; convención de nombres; versionado semántico (SemVer) y tags firmados para releases; CHANGELOG generado (p. ej., Conventional Commits).
- Revisiones y calidad de código: Merge Requests obligatorias para todo cambio (prohibidos commits directos a main); ≥ 1 revisor independiente; políticas “no fast-forward”; linters y estándares de estilo; cobertura mínima de pruebas ≥ 70 % (ajustable por la Agencia).
- CI/CD en GitLab: pipeline con etapas build → test → security → package → deploy; ambientes dev, test, prod; aprobaciones manuales para deploy a prod; rollback documentado; artefactos firmados y retenidos.
- Seguridad en el ciclo: SAST/DAST/Dependency Scanning activados; gestión de secretos via variables protegidas; políticas de firma de commits/tags; escaneo de imágenes de contenedor.
- Gestión de issues y trazabilidad: uso de Issues/Epics/Milestones y tableros Kanban; etiquetado por componente/trámite; vínculo Issue ↔ MR ↔ Commit ↔ Release; referencia a requisitos, casos de uso y ADRs.
- Documentación en el repo: README y CONTRIBUTING; plantillas de Issue/MR; carpeta /docs con arc42/C4; carpeta /adr para decisiones de arquitectura; guía de ramas, commits y releases.
- Observabilidad del proceso: retención de logs de pipeline, métricas de tiempos de build/test/deploy y trazabilidad commit→build→deploy.

- Criterios de aceptación del flujo: 100 % de los cambios ingresan por MR; 0 % de commits directos a main; 100 % de pipelines verdes antes de merge; releases taggeadas y versionadas; evidencias de revisión y aprobaciones registradas.

Pendientes institucionales (se completarán al inicio): estructura de repositorios, permisos finos, convenciones definitivas (branching, versionado, normas de commit) y credenciales de acceso al GitLab de la Agencia.

Nota: Durante el inicio del desarrollo no será necesario aplicar todas estas recomendaciones – buenas prácticas, las mismas se irán implementando en conjunto con el equipo de desarrollo.

3 CONTEXTO

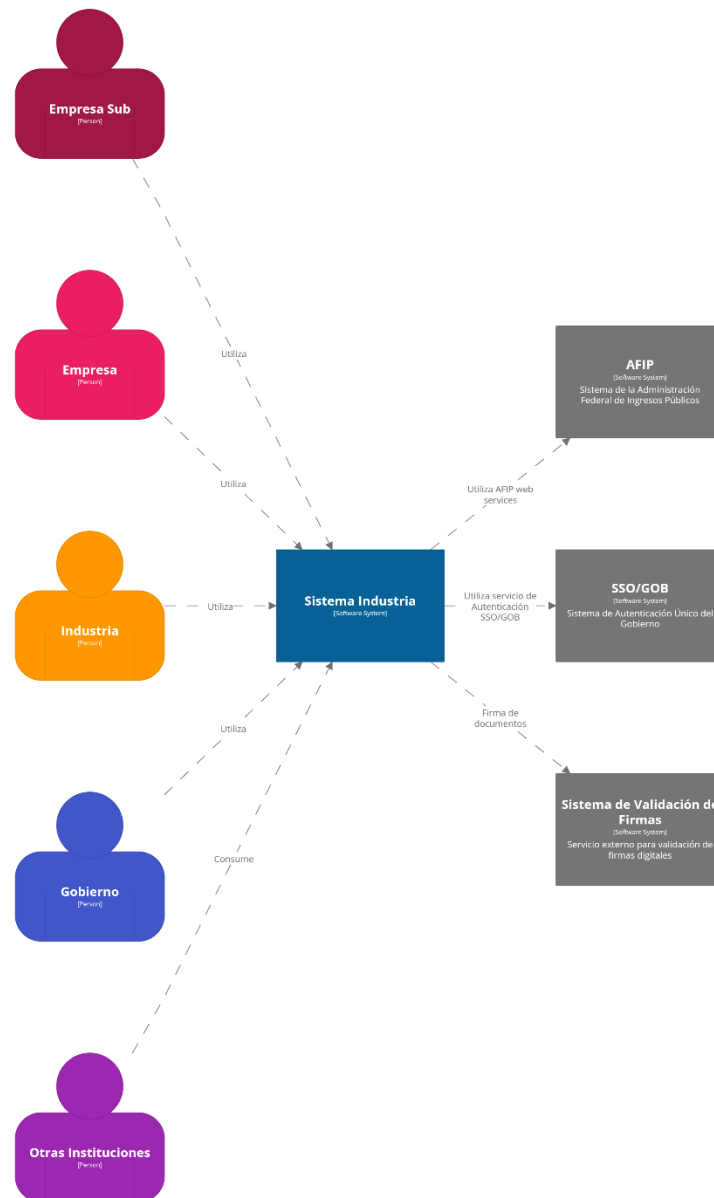


Figura 1: Vista de contexto general del sistema

3.1 VISIÓN GENERAL Y OBJETIVOS

Este prototipo de sistema tiene como propósito centralizar y automatizar la gestión de trámites de la Secretaría de Industria provincial, garantizando trazabilidad, auditoría y flexibilidad ante flujos estándar y excepciones.

3.1.1 Actores principales

- **Empresas** (subrégimen y no subrégimen): inician y dan seguimiento a sus trámites.
- **Secretaría de Industria**: diversas direcciones y roles responsables de validar y controlar cada trámite, dándole avance o no según corresponda.
- **Agentes externos**: otras oficinas gubernamentales que validan información específica.

3.2 OBJETIVOS CLAVE

1. Modelar cada trámite como una máquina de estados configurable.
2. Registrar todas las operaciones con auditoría completa, garantizando la integridad de la información mediante firmas digitales.
3. Permitir flujos estándar y desviaciones/excepciones documentadas.
4. Proveer panel de KPIs y exportación de historiales.
5. Registrar y firmar internamente todas las acciones realizadas por usuarios en trámites y operaciones sensibles, incluyendo las firmas digitales externas cuando corresponda.
6. Garantizar la posibilidad de desplegar el sistema tanto en servidores propios (on premise) como en infraestructura cloud, utilizando contenedores.

4 ESTRATEGIA DE SOLUCIÓN

La arquitectura del Sistema Integral de Gestión de Trámites (SIGT) se sustenta en decisiones estratégicas orientadas a garantizar trazabilidad, confiabilidad, seguridad y mantenibilidad, en alineación con los requisitos del nuevo Sistema de Gestión de Calidad (SGC) y estándares internacionales (ISO 9001, 25010, 27001, OWASP ASVS, entre otros).

4.1 DECISIONES TECNOLÓGICAS

- Se opta por una arquitectura modular, basada en contenedores (Docker o Podman), que permite el despliegue tanto en entornos on premise como en la nube, según disponibilidad.
- Las interfaces estarán construidas siguiendo principios RESTful y utilizando intercambio de datos en formato JSON, facilitando la interoperabilidad con sistemas externos (AREF, ARCA, SGC).
- Se emplearán mecanismos de autenticación estándar (OpenID Connect) y cifrado de datos en tránsito y en reposo para cumplir con requisitos de seguridad y confidencialidad.

4.2 DESCOMPOSICIÓN DEL SISTEMA

- Cada trámite será modelado como una máquina de estados configurable, lo que permite flexibilidad y adaptación a nuevas normativas sin necesidad de reescritura del código.

- El diseño se basa en la separación de responsabilidades entre presentación, lógica de negocio y persistencia, utilizando patrones de diseño bien conocidos.

4.3 LOGRO DE OBJETIVOS DE CALIDAD

- La trazabilidad y auditabilidad se aseguran mediante registro completo de operaciones, firmas digitales y generación de historiales exportables.
- La mantenibilidad se aborda con documentación exhaustiva, automatización de despliegues y una arquitectura clara que permite transferencia tecnológica efectiva.
- El rendimiento será garantizado mediante pruebas de carga para asegurar tiempos de respuesta <2s con al menos 50 usuarios concurrentes.

4.4 DECISIONES ORGANIZACIONALES

- Se propone el uso de *tecnologías maduras y ampliamente adoptadas*, con preferencia por soluciones open source auditables.
- Las integraciones con sistemas externos serán progresivas, evaluando factibilidad técnica y relevancia funcional en función de casos de uso.
- Se considera desde el inicio la capacitación y eventual transferencia a equipos técnicos del gobierno provincial, priorizando la autonomía operativa futura.

Estas decisiones forman la base sobre la cual se construirá el resto de la arquitectura y guiarán las decisiones técnicas detalladas en las siguientes secciones.

5 COMPONENTES

Se propone un sistema monolítico principal asilado como contenedor (Docker), con separación en contenedores específicos para servicios de logs, auditoría y mantenimiento, y con módulos internos adicionales para nuevas funcionalidades.

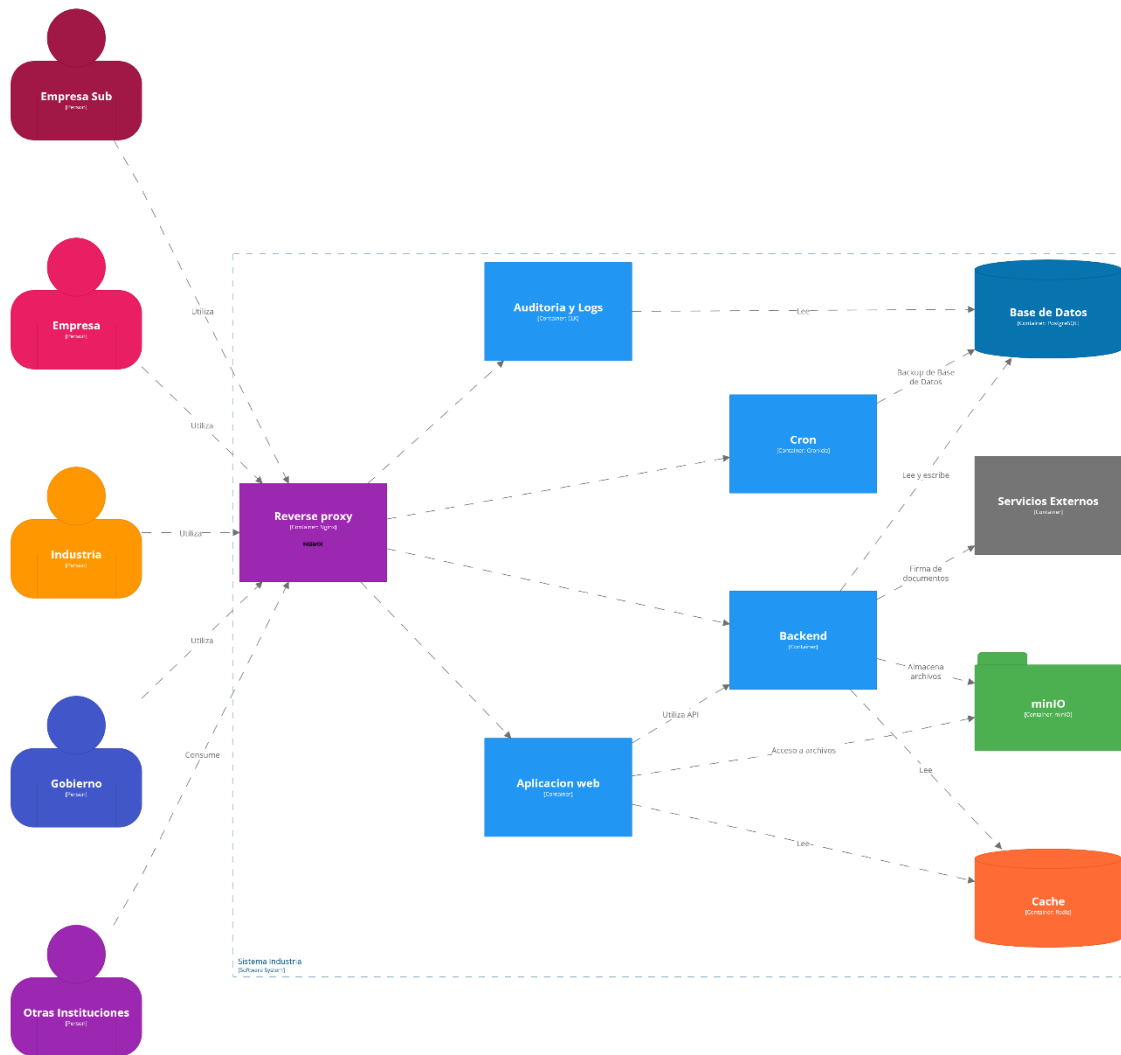


Figura 2: Vista de componentes principales del sistema

5.1 FLUJO DE INTERACCIÓN GENERAL

Las Empresas (usuarios externos) y la Secretaría de Industria (usuarios internos de Gobierno) acceden a la Aplicación Web a través del Reverse Proxy. La Aplicación Web se comunica con el Backend para todas las operaciones de negocio. El Backend interactúa con la Base de Datos para persistencia, con minIO para almacenamiento de archivos y con la Cache para optimizar el rendimiento. Las acciones importantes se registran en el Servicio de Auditoría y Logs con eventos JSON y correlationId. El sistema se integra con servicios externos como AFIP y el Sistema de Validación/Firma para funcionalidades específicas de los trámites. Las tareas programadas son gestionadas por el servicio Cron.

5.2 COMPONENTES PRINCIPALES Y FUNCIONES

1. Frontend (SPA):
 - a. Framework (React, Angular o Vue).
 - b. Autenticación (OAuth2/JWT) y control de permisos dinámico.
 - c. Vista de formularios de trámite, seguimiento de estado y panel de excepciones.
2. API REST (Monolito Principal):
 - a. Implementación basada en arquitectura RESTful.
 - b. Encargado de exponer endpoints para gestión de trámites, usuarios, firmas, exportaciones, dashboard, etc.
 - c. Control de autenticación, autorización y enrutamiento lógico interno.
 - d. Enrutamiento a servicios externos y otros módulos.
 - e. Autorización por rol.
 - f. Control de versionado de API.
3. Servicio de Usuario y Autenticación:
 - a. Gestión de perfiles, roles y permisos.
 - b. Registro de empresas y proceso de onboarding auto-gestionado.
4. Orquestador / BPM Engine:
 - a. Componente encargado de modelar y ejecutar los flujos de los distintos trámites, como máquinas de estados configurables.
 - b. Administra transiciones, validaciones, asignaciones y tareas manuales o automáticas dentro del ciclo de vida de cada trámite.
 - c. Puede implementarse de dos maneras:
 - i. Opción A: Uso de BPM open source externo: como Camunda, Zeebe, Flowable, Joget o Node-RED (en el ecosistema Node.js), desplegado como componente embebido o conectado vía API.
 - ii. Opción B: Motor propio dentro del monolito: desarrollado a medida para ajustarse a las necesidades específicas del sistema, manteniendo control completo sobre reglas, validaciones y versionado.
 - d. Requiere de un editor de flujos para el diseño visual de estados y transiciones (propio o embebido del motor elegido).
 - e. Soporta excepciones y desvíos documentados, versionado de procesos y trazabilidad de cada instancia.
5. Contenedor de Logs, Auditoría y Mantenimiento:
 - a. Servicio desacoplado del monolito, ejecutado en su propio contenedor.
 - b. Administra logs del sistema, auditoría de operaciones y tareas de mantenimiento.
 - c. Utiliza el stack ELK (Elasticsearch, Logstash, Kibana) para la gestión de logs.
 - d. Framework propuesto para auditoría: Audit.NET (si stack .NET) o Spring Boot Actuator con AOP (si stack Java).
 - e. Control de definiciones de cron jobs, backups automáticos de base de datos y mantenimiento periódico.
 - f. Posee interfaz de administración específica para la gestión de estas funciones.
 - g. Registro inmutable de eventos (Event Store o append-only log).
 - h. Trazabilidad completa: quién, cuándo, qué operación.
6. Base de Datos:
 - a. Relacional (PostgreSQL/MySQL) con tablas de historial y versionado de requisitos.

- b. Mecanismo de auditoría (row versioning o tablas auxiliares).
- 7. Servicio de Firma Digital Remota:
 - a. Integración con proveedores de firma remota delegada: API Nación y/o proveedor privado.
- 8. Servicio de Exportación y Reportes (Integrado):
 - a. Módulo dentro de la aplicación principal.
 - b. Generación de documentos estándar (PDF/Word) con historial completo.
 - c. Plantillas configurables para exportación.
 - i. Generación de documentos estándar (PDF/Word) con historial completo.
 - ii. Plantillas configurables.
- 9. Dashboard y BI (Integrado):
 - a. Incluido dentro de la aplicación principal.
 - b. Visualización de KPIs en tiempo real.
 - c. Filtros por tipo de trámite, estado, tiempos medios, cuellos de botella.
 - d. Widgets configurables por perfil de usuario.
- 10. Módulo de Notificaciones por Correo (Integrado):
 - a. Envía notificaciones automáticas a los actores involucrados cuando cambian los estados de los trámites.
 - b. Configuración de eventos notificables, destinatarios y contenido por parte de administradores.
- 11. Módulo de Comunicaciones y Mesa de Entradas (Integrado):
 - a. Simula la recepción y envío de notas oficiales (internas y externas).
 - b. Algunas comunicaciones estarán asociadas a trámites específicos, otras serán generales.
 - c. Incluye trazabilidad, archivado y relación con entidades externas.
- 12. Integración con Sistema de Actas de la CAAE (Integrado):
 - a. Permite asignar trámites a futuras reuniones de la comisión.
 - b. Exportación de trámites para tratamiento en comisión.
 - c. Registro y actualización del dictamen posterior a la reunión.
- 13. Contenedor de Cache: Redis (instancia local o Upstash) para almacenamiento temporal de datos, reducción de latencia y mejora de performance en operaciones frecuentes.
- 14. Contenedor de Almacenamiento de Archivos: servicio tipo S3 (MinIO en contenedor dedicado) para guardar documentación adjunta, formularios y archivos firmados digitalmente.
- 15. Contenedor de datos: Base de datos relacional SQL para almacenar toda la información de la aplicación, excluyendo archivos.

5.3 FLUJO DE UN TRÁMITE (EJEMPLO)

Inicio: Empresa completa formulario de inicio.

Validación inicial: Reglas automáticas (campos obligatorios, formato).

Asignación: Dirección o agente responsable revisa manualmente.

Transición de estado: Aprobado / Rechazado / Pendiente.

Excepción (opcional): Desvío documentado con motivos y fecha.

Firma de documentos: Integración con servicio de firma.

Finalización: Generación de certificado y exportación.

6 MODULOS

6.1 FRONTEND

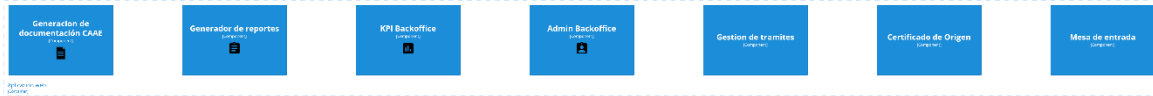


Figura 3: Vista de detalle del componente "Frontend"

El frontend se estructura como una aplicación web de página única (SPA), desarrollada en un framework moderno como React, Angular o Vue. Entre sus componentes principales se incluyen:

- **Generación de documentación CAAE, Generador de reportes y KPI Backoffice:** Estos módulos permiten a los usuarios acceder a reportes, métricas clave (KPIs) y documentos exportables, integrando plantillas configurables y registros históricos.
- **Admin Backoffice y Gestión de trámites:** Proveen funcionalidades para la gestión administrativa de los trámites, su seguimiento y control de excepciones, siendo el canal principal de interacción para los actores del sistema.
- **Certificado de Origen y Mesa de entrada:** Implementan vistas específicas para gestionar la emisión de certificados y comunicaciones institucionales, con trazabilidad y asociación a trámites particulares.

Todos estos módulos se integran con mecanismos de autenticación OAuth2/JWT, control dinámico de permisos (RBAC + ABAC), y consumen servicios REST del backend para operaciones seguras y centralizadas.

6.2 BACKEND



Figura 4: Vista de detalle de componente "Backend"

El backend del sistema sigue una arquitectura basada en módulos desacoplados y bien definidos:

- **Backoffice de administración:** Centraliza funciones administrativas para la configuración de parámetros, perfiles y tareas de mantenimiento general del sistema.
- **Notificador de eventos:** Encargado del envío de notificaciones automáticas por correo u otros canales, en función de eventos clave en el ciclo de vida de los trámites.
- **Generador de documentación y reportes:** Módulo integrado que produce documentos exportables (PDF/Word), certificados y reportes con historial completo de cambios, utilizando plantillas configurables.
- **Certificado de Origen:** Componente que gestiona la emisión, validación y trazabilidad de certificados, integrando lógica de negocio específica y firmas digitales.
- **Motor de trámites:** Representa el orquestador central del sistema, modelando y ejecutando flujos de trabajo como máquinas de estado. Permite gestionar validaciones, transiciones, tareas manuales/automáticas y excepciones dentro del ciclo de vida de cada trámite.

Cada uno de estos módulos se posee autenticación/autorización, control de versiones y exposición segura de servicios. Además, el sistema se complementa con una base de datos relacional versionada, un sistema de auditoría basado en logs inmutables, y un contenedor especializado en tareas de mantenimiento y monitoreo del estado del sistema.

6.3 MOTOR DE TRAMITES

El módulo de motor de flujos o BPM Engine es una pieza central del sistema, ya que administra la lógica que permite que cada trámite se ejecute como una máquina de estados configurable y trazable. Este motor debe cumplir con los siguientes requerimientos:

6.3.1 Requisitos específicos del motor de flujo:

1. Definición de flujos y estados:
 - a. Cada tipo de trámite debe contar con una definición editable de su flujo como una máquina de estados.
 - b. Los estados representan pasos lógicos del trámite y deben incluir metadatos: nombre, descripción, tipo (inicio, revisión, firma, finalización, etc.).
2. Transiciones y condiciones:
 - a. Las transiciones entre estados deben poder configurarse en función de condiciones específicas.
 - b. Estas condiciones pueden incluir: cumplimiento de requisitos, firmas completadas, evaluaciones manuales, resultados automáticos, vencimientos o validaciones externas.
3. Gestión de requisitos por estado:
 - a. Cada estado puede requerir un conjunto de requisitos específicos (documentación, formularios, validaciones).
 - b. Estos requisitos deben poder modificarse manteniendo el historial (versionado).
4. Asignación de responsabilidades:

- a. Debe definirse quién (qué rol o usuario) es responsable de actuar en cada estado.
 - b. El motor debe gestionar notificaciones automáticas y tareas asignadas por actor.
- 5. Control de desvíos y excepciones:
 - a. El flujo debe contemplar caminos alternativos para gestionar excepciones, interrupciones y reanudaciones documentadas.
 - b. Cada excepción debe quedar registrada con metadatos: motivo, fecha, actor responsable y paso que la originó.
- 6. Finalización e interrupción:
 - a. El motor debe permitir marcar un trámite como finalizado de forma estándar o por resolución excepcional.
 - b. También debe contemplarse la posibilidad de interrupción del trámite por causa justificada.
- 7. Historial y trazabilidad completa:
 - a. Cada transición de estado debe generar un evento auditable con fecha, actor, condiciones evaluadas y acción realizada.
 - b. El historial debe poder reconstruirse en forma cronológica o por agrupamiento lógico.
- 8. Interfaz de modelado (opcional):
 - a. Si se utiliza un BPM externo, se recomienda que cuente con interfaz visual para definir y versionar flujos.
 - b. Si el motor es interno, se puede desarrollar una interfaz simplificada para administración de estados, transiciones y reglas.
- 9. Soporte para plantillas y reutilización:
 - a. Las definiciones de flujo deben poder reutilizarse como plantillas base para otros trámites similares.
 - b. Se debe permitir clonar, versionar y adaptar flujos a partir de uno existente.
- 10. Integración con el resto del sistema:
 - a. El motor debe integrarse con el módulo de requisitos, de validaciones automáticas, firma digital, exportación e informes.
 - b. Debe exponer una interfaz interna clara (API o servicios embebidos) para que otros módulos puedan consultar o disparar acciones del flujo.

6.4 AUTENTICACIÓN Y AUTORIZACIÓN

6.4.1 Autenticación y gestión de identidades (flujo diferenciado):

- 1. Usuarios internos (Gobierno): la autenticación se realiza mediante el SSO de la Agencia (OAuth2/OIDC). El Reverse Proxy redirige al proveedor SSO para el login; una vez autenticado, la Aplicación Web recibe el token y lo envía al Backend en cada solicitud. El Backend valida el token (firma, vigencia, scopes/roles) y autoriza la operación según el modelo de autorización (RBAC/ABAC).
- 2. Usuarios externos (Empresas): la autenticación es gestionada por el sistema con credenciales locales. Se proveen registro, recuperación de contraseña y políticas de seguridad (complejidad, expiración y bloqueo por intentos fallidos). El token de sesión/API es emitido por el Backend tras validar las credenciales. Se evaluará la incorporación de 2FA (p. ej., TOTP) para fortalecer el acceso externo.

6.4.2 Secuencia resumida de interacción por tipo de usuario:

- **Interno (SSO):** Navegador → Reverse Proxy → Redirección SSO → (Login SSO) → Redirección a UI con token → UI → Backend (token) → Backend ↔ DB/Cache/minIO → Backend ↔ AFIP/Firma → Auditoría/Logs.
- **Externo (local):** Navegador → Reverse Proxy → UI (form login/registro) → Backend (validaciones credenciales / emisión token) → Backend ↔ DB/Cache/minIO → Backend ↔ AFIP/Firma → Auditoría/Logs.
En ambos casos, **Cron** ejecuta procesos diferidos (notificaciones, conciliaciones, reintentos de firma, mantenimiento de índices), registrando resultados en **Auditoría/Logs**.

6.4.3 Consideraciones de seguridad aplicadas al flujo:

- **Tokens:** validación de firma/expiración y **scopes/roles** en cada request al Backend.
- **MFA (evaluación):** para externos, habilitación opcional de **2FA**; para internos, se respeta la política del **SSO de la Agencia**.
- **Trazabilidad:** todas las solicitudes críticas incluyen **correlationId**, con logs **JSON** para diagnóstico y auditoría.
- **Límites de firma:** cuando correspondan operaciones de **firma/validación** de documentos, el Backend aplica **segmentación/optimización** si se superan límites de tamaño o páginas del servicio nacional.

6.5 SISTEMA DE FIRMA DIGITAL REMOTA

- Integración con proveedores de firma remota delegada: API Nación y/o proveedor privado.
- SDK/Librería propia o de terceros para firmar, validar y sellar documentos (soporte PDF/PDF-A).
- Gestión de límites del proveedor (tamaño en MB / cantidad de páginas): segmentación automática y optimización de PDF.
- Validación: verificación de OCSP/CRL, cadena de certificados, y hash pre/post firma (trazabilidad probatoria).
- Sellado de tiempo (si el proveedor lo ofrece) y embebido de evidencias en el documento resultante.
- Auditoría: registro JSON con correlationId, IDs de transacción del proveedor, métricas de latencia y resultado.
- Plantillas y parámetros de firma configurables (motivo, ubicación, posiciones/visibilidad de sello, firmante).
- Interfaz REST interna: endpoints POST /firmas/solicitar, GET /firmas/{id}/estado, POST /firmas/validar.
- Almacenamiento de documentos firmados en MinIO (S3) y metadatos en BD relacional.
- Seguridad: manejo de credenciales/keys del proveedor vía secretos del entorno o credenciales ofuscadas en base de datos; tokenización de llamadas.

- Monitoreo: métricas (tasa de éxito, tiempo medio de firma, ratio de segmentación), alertas por error/timeout.

6.6 CUMPLIMIENTO DE REQUISITOS FUNCIONALES

A continuación, se detalla cómo cada característica del sistema garantiza el cumplimiento de los requisitos funcionales definidos:

Requisito funcional	Componente(s) Responsable(s)
1. Trazabilidad de todas las operaciones	Servicio de Logs y Auditoría: registro inmutable de eventos (append-only). Base de Datos: tablas auxiliares de historial y versionado.
2. Registro auditable	Servicio de Logs y Auditoría: captura metadatos (quién, cuándo, qué). API Gateway: intercepta y registra llamadas para asegurar trazabilidad completa.
3. Roles y permisos editables por administrador	Servicio de Usuario y Autenticación: gestión de perfiles, roles y permisos dinámicos; interfaz de administración para CRUD de roles.
4. Requisitos de inicio, continuación y finalización editables sin perder trazabilidad	Base de Datos: versionado de esquemas de requisitos. Orquestador/BPM Engine: permite definir y actualizar flujos y validaciones; cada cambio genera una nueva versión de proceso sin alterar historiales previos.
5. Modelado de trámites como máquinas de estados con revisiones automáticas o manuales	Orquestador/BPM Engine: motor de estado configurable, reglas de validación automáticas y tareas manuales que determinan transiciones.
6. Gestión de excepciones y desviaciones documentadas	Orquestador/BPM Engine: módulo de excepciones para rutas alternas. Servicio de Logs y Auditoría: registro detallado de cada excepción, motivos y fechas.
7. Verificación fácil del estado, actores involucrados, responsables, desvíos, fechas, etc.	Frontend (SPA): panel de seguimiento con vista de historial y línea de tiempo. Dashboard/BI: reportes y filtros para explorar información detallada por trámite.
8. Integración con firma digital	Servicio de Firma Digital: APIs para solicitar y validar firmas, notificaciones de firma completada. API Gateway: orquesta peticiones hacia proveedor de firma.
9. Exportación de estado e historial a documento estándar	Servicio de Exportación y Reportes: genera PDF/Word con plantillas que incluyen cronología completa.
10. Dashboard de seguimiento con KPI configurables	Dashboard/BI: widgets configurables, métricas en tiempo real, alertas y notificaciones.
11. Auto-gestión de perfiles y permisos por usuarios dentro de límites	Servicio de Usuario y Autenticación: interfaz de usuario para autoadministración de perfiles y solicitud de cambios de permiso, sujeto a aprobación administrativa.
12. Onboarding auto gestionado de empresas con verificación posterior	Frontend (SPA) + Servicio de Usuario y Autenticación: flujo de registro guiado, validaciones automáticas iniciales. Orquestador/BPM Engine: tarea manual de verificación final por parte de Secretaría.

13. Rol de administrador para gestión de permisos, roles y configuraciones	Servicio de Usuario y Autenticación: perfil de administrador con privilegios para CRUD de roles, permisos, plantillas de trámite y KPI.
14. Vinculación con actas de la CAAE	Orquestador/BPM Engine: permite asignar trámites a reuniones futuras y registrar dictámenes. Servicio de Exportación: generación de documentos para revisión en comisión. Base de Datos: asociación de trámites a eventos de la CAAE.
15. Gestión de envío y recepción de notas oficiales (mesa de entrada/salida)	Módulo de Comunicaciones (integrado al monolito): gestiona envíos/recepciones, relación con trámites, archivo y trazabilidad de comunicaciones.
16. Notificaciones automáticas por correo electrónico	Servicio de Notificaciones: envío de alertas por cambio de estado. Configuración por administrador para definir qué eventos generan notificación, a qué actores, y con qué contenido.
17. Firma de operaciones internas y externas, con trazabilidad completa	Servicio de Logs y Auditoría: registro de todas las acciones del usuario. Servicio de Firma Digital: registro de documentos firmados digitalmente (externa) o por identidad interna del usuario autenticado.
18. Despliegue en entornos on-premise o nube	Docker + Docker Compose: portabilidad total del sistema, adaptable a servidores propios del gobierno o infraestructura cloud.

Tabla 6: Relacion de componentes y requisitos funcionales principales

7 ARQUITECTURA AMPLIADA – MOTOR DE TRAMITES

7.1 PROPÓSITO Y ALCANCE

El Motor de Trámites es el componente responsable de definir, versionar y ejecutar los flujos de trabajo de los distintos tipos de trámite. Para ello:

- Interpreta la definición del flujo (plantilla) y gobierna la instancia en ejecución.
- Garantiza la trazabilidad completa de transiciones, acciones y decisiones.
- Orquesta la integración con validaciones, notificaciones, firma digital y generación documental (módulos del backend).

En el modelo de datos del SITDF, el núcleo se organiza alrededor de Tipo de Trámite (plantilla/modelo) y Trámite (instancia concreta). La separación plantilla/instancia permite versionado histórico y compatibilidad hacia atrás (instancias iniciadas no se ven afectadas por cambios posteriores).

7.2 COMPOSICIÓN INTERNA DEL MÓDULO

Para soportar un flujo altamente dinámico (nodos, datos y reglas configurables), se propone descomponer el Motor en subcomponentes internos (en forma de servicios internos o capas dentro del monolito):

1. Repositorio de Plantillas (Template Store)
 - Almacena definiciones versionadas del flujo (grafos de nodos y conexiones), formularios y reglas.
 - Mantiene metadatos: nombre, descripción, vigencia, estado (borrador/publicado/deprecado), roles administradores, etc.
 - Cada modificación crea una nueva versión de la plantilla sin alterar historiales previos.
2. Runtime / Ejecutor de Flujos (Flow Runtime)
 - Interpreta la plantilla versionada y ejecuta la instancia como máquina de estados.
 - Coordina handlers por tipo de nodo (entrada/condición/procesamiento/acción manual/asignación/inicio/fin).
 - Dispara transiciones automáticas y crea tareas pendientes para pasos manuales.
3. Gestor de Tareas Manuales (Worklist / Task Manager)
 - Materializa “pendientes” por rol/usuario/sector.
 - Administra asignación, reasignación, vencimientos y estados de tarea (pendiente/en curso/resuelta/observada).
4. Motor de Reglas y Expresiones (Rules/Expression Engine)
 - Evalúa condiciones sobre datos cargados en la instancia + datos maestros (empresa/producto/proceso).
 - Debe soportar reglas configurables por plantilla y versionadas.
5. Gestor de Excepciones y Excepciones (Exception Manager)

- Administra desvíos documentados, rutas alternativas y bypass de nodos, registrando motivo/fecha/actor/paso de origen.
6. Auditoría y Firma de Acciones (Audit + Signature Adapter)
 - Registra eventos inmutables con autoría, timestamps, condiciones evaluadas y acción realizada.
 - Integra con servicios de auditoría/logs y con firma digital (según corresponda: firma interna y/o firma digital remota).
 7. Integradores internos
 - Notificaciones (event-driven por cambios de estado).
 - Documentos/Reportes (generación de PDFs/Word con cronología completa).
 - Módulo de permisos (RBAC/ABAC) para autorizar acciones en cada nodo/estado.

7.3 MODELO DE FLUJO: GRAFO DE NODOS CONECTADOS

Cada Tipo de Trámite se define como un grafo dirigido (nodos + conexiones), con un único nodo Inicio y múltiples nodos Fin (finalizaciones alternativas), manteniendo la semántica de máquina de estados configurable.

- **Nodos:** representan pasos del trámite (equivalentes a estados lógicos) y deben incluir metadatos configurables (nombre, descripción, tipo, responsables, condiciones de avance, requisitos, etc.).
- **Conexiones / Transiciones:** unen nodos e incorporan reglas/condiciones de salto. Las transiciones deben configurarse por condiciones específicas, incluyendo cumplimiento de requisitos, firmas, evaluaciones manuales, resultados automáticos y validaciones externas.

7.4 TIPOS DE NODOS Y SU INTERACCIÓN

El runtime resuelve cada nodo mediante un handler específico por tipo. Los tipos principales y su comportamiento:

1. Nodo de Entrada de Datos (Formulario)

- Presenta un formulario con campos tipados (texto, numérico, select, fechas, adjuntos, etc.).
- Al guardar: valida esquema, persiste datos y genera evento auditable (quién, cuándo, qué cambió).
- Puede definir “requeridos”, reglas de consistencia, y dependencias hacia nodos de condición/procesamiento.

2. Nodo de Condición (Evaluación automática)

- Se evalúa automáticamente al ingresar o cuando cambian datos dependientes.
- Toma como insumo:
 - Datos cargados hasta el momento (instancia).
 - Datos maestros: empresa, producto, proceso (según aplique).
 - Parámetros y reglas del nodo (plantilla versionada).
- Registra resultado (true/false/múltiples salidas) y transiciona sin intervención humana.

3. **Nodo de Procesamiento (Acción automática)**

- Ejecuta acciones automáticas: guardar estados, generar documentación, enviar notificaciones, actualizar relaciones (producto/proceso), etc.
- Puede incluir “acciones preconfiguradas” (catálogo), para estandarizar comportamientos y minimizar lógica ad-hoc.
- Debe ser idempotente y auditable: cada ejecución genera evento con resultado y artefactos producidos.

4. **Nodo de Acción Manual**

- Crea una tarea asignada a rol/usuario/sector (Secretaría de Industria) o a la empresa o algún agente externo.
- Hasta completarse, el trámite queda en estado de espera.
- Al completarse: registra evidencia (adjuntos, observaciones, decisión) y firma/autoría.

5. **Nodo de Asignación de Usuario / Sector**

- Transfiere el “ownership operativo” del trámite a otro sector/rol/usuario (interno o externo).
- Puede crear automáticamente la tarea de recepción/aceptación y notificar.

6. **Nodos especiales: Inicio y Fin**

- Inicio: instancia el trámite (selecciona versión de plantilla, setea responsables y contexto).
- Fin: cierra con estado final (p. ej., “finalizado”, “finalizado con excepciones”, “interrumpido”), generando documentación final y registro definitivo.

7.5 ASIGNACIÓN DE RESPONSABILIDADES Y RESPONSABLE MÁXIMO

El motor debe permitir definir quién (rol/usuario) es responsable de actuar en cada estado, y gestionar notificaciones automáticas/tareas asignadas por actor.

Se incorporan dos niveles de responsabilidad:

- **Responsable máximo del trámite** (siempre asignado; dueño funcional global).
- **Responsable del paso/nodo** (dinámico, según nodo y sector).

Ambos deben figurar en el estado visible del trámite (“Paso X – a cargo de Y”), manteniendo coherencia con el tablero y el historial.

7.6 ESTADOS GENÉRICOS DEL TRÁMITE (VISTA EXTERNA E INTERNA)

A nivel de UX y reporting, se define un conjunto de estados genéricos comunes:

- Inicio
- En proceso (con detalle: paso actual y actor responsable)
- En revisión (con detalle: sector revisor)
- Esperando rectificación

- Esperando confirmación / operación manual
- Finalizado
- Finalizado con excepciones

Implementación recomendada:

- Mantener un estado global (catálogo acotado, transversal) y un estado técnico (nodo actual + subestado).
- El motor actualiza ambos, y cada transición genera evento auditable reconstruible.

7.7 EXCEPCIONES Y EXCEPTUACIÓN DE NODOS

El flujo debe contemplar caminos alternativos para excepciones, interrupciones y reanudaciones documentadas, registrando motivo, fecha, actor responsable y paso originante.

Adicionalmente, se incorpora la capacidad de exceptuar nodos (bypass controlado) con reglas específicas:

7.7.1 Modelo de exceptuación por nodo

Cada nodo define:

- Condiciones de elegibilidad de exceptuación (p. ej., “campo faltante”, “documento no disponible”, “validación externa caída”).
- Requisitos de aprobación: rol(es) habilitados, posibilidad de doble control, obligatoriedad de justificación, adjuntos requeridos, etc.
- Impacto funcional: qué reglas quedan “deshabilitadas” o cómo se reinterpretan condiciones dependientes (p. ej., “considerar como ‘waived’ en evaluaciones posteriores”).

7.7.2 Transacción de excepción auditable

La acción de exceptuar debe:

- Requerir autenticación/autorización.
- Generar un evento inmutable con: nodo, datos afectados, justificación, aprobador(es), timestamp y evidencia.
- Marcar el trámite para que el cierre pueda reflejar “finalizado con excepciones” cuando corresponda.

7.8 AUDITORÍA, FIRMA Y RECONSTRUCCIÓN COMPLETA DEL HISTORIAL

Toda transición y acción (automática o manual) debe generar un evento auditable con fecha, actor, condiciones evaluadas y acción realizada. El historial debe reconstruirse cronológicamente o por agrupamiento lógico.

7.8.1 Recomendación de arquitectura de auditoría

- Implementar un **registro inmutable de eventos (append-only / event store)** para el motor, alineado con la arquitectura de auditoría y trazabilidad del sistema.

- Incluir correlationId/trazas en logs estructurados para observabilidad end-to-end en operaciones críticas.

7.8.2 Firma

- Para acciones internas: firma “por identidad autenticada” registrada en auditoría.
- Para documentos/acciones que lo requieran: integración con **Servicio de Firma Digital** (firma remota) y su registro asociado.

7.9 VERSIONADO DE PLANTILLAS Y COMPATIBILIDAD HACIA ATRÁS

Cada modificación del flujo (plantilla) debe generar una nueva versión. La instancia del trámite referencia de forma inmutable la versión con la que fue iniciada, evitando que cambios posteriores afecten trámites en curso.

Recomendación de operación:

- Estados de publicación: **Borrador** → **Publicado** → **Deprecado**.
- Solo versiones publicadas pueden iniciar trámites nuevos.
- Se permite clonar una plantilla existente para acelerar la creación/variación de un trámite (reutilización), manteniendo versionado.

7.10 PERSISTENCIA DE DATOS DINÁMICOS (PLANTILLA E INSTANCIA)

Dado el carácter altamente dinámico:

- La plantilla y la instancia deben soportar estructuras flexibles (p. ej. JSON) para definición de nodos, formularios, reglas y estado.
- Se recomienda complementar con tipos/estructuras de código (DTOs/validadores) para forzar esquemas y asegurar calidad de datos.

Estrategia práctica (híbrida JSON + consultas)

- Guardar:
 - Definición de flujo: template_definition (JSON).
 - Estado de instancia: instance_state (JSON).
 - Datos de formularios: instance_data (JSON por versión/esquema).
- Para reporting/condiciones intensivas: mantener un “índice” **EAV** o materializaciones para atributos clave consultables, sin perder la flexibilidad del payload principal.

7.11 INTERFAZ DE MODELADO Y ALTERNATIVA DE IMPLEMENTACIÓN (BPM EXTERNO VS MOTOR PROPIO)

El documento contempla dos enfoques:

- **Opción A:** BPM open source externo (Camunda/Zeebe/Flowable/Joget/Node-RED), embebido o vía API.

- **Opción B:** motor propio dentro del monolito, a medida, con editor simplificado.

Independientemente de la opción, el módulo debe:

- Disponer de un editor de flujos (visual o simplificado) para definir y versionar estados/transiciones.
- Exponer una interfaz interna clara (API/servicios embebidos) para que otros módulos consulten o disparen acciones del flujo.

7.12 FLUJOS OPERATIVOS CLAVE

7.12.1 Inicio de un trámite

1. Selección de Tipo de Trámite (versión publicada).
2. Creación de instancia: se asocia a Empresa y, según tipo, a Producto/Proceso.
3. Se asigna responsable máximo.
4. Se ingresa al nodo Inicio y se crea el primer nodo operativo.
5. Se registra evento “Trámite creado”.

7.12.2 Ejecución de nodos automáticos (condición/procesamiento)

1. Runtime detecta nodo automático.
2. Recolecta contexto (instance_data + maestros).
3. Evalúa reglas/ejecuta acción.
4. Persiste resultado y registra evento.
5. Determina transición y avanza.

7.12.3 Ejecución de nodos manuales

1. Runtime crea Task (worklist) para rol/usuario/sector.
2. Estado global pasa a “Esperando confirmación / operación manual”.
3. Actor ejecuta la acción (adjunta, revisa, aprueba/rechaza, etc.).
4. Se registra evento firmado (autoría) y se avanza al siguiente nodo.

7.12.4 Excepción

1. Actor solicita exceptuar nodo (con justificativo/evidencia).
2. Sistema valida roles y requisitos del nodo.
3. Si corresponde aprobación: se genera tarea de aprobación.
4. Al aprobar: se marca nodo como “exceptuado”, se registran eventos, y el runtime recalcula condiciones dependientes con semántica “waived”.
5. Si el cierre ocurre con excepciones vigentes, el fin puede ser “finalizado con excepciones”.

7.12.5 Clonado (para empresas)

- Se crea nueva instancia a partir de un trámite previo, copiando únicamente datos iniciales configurados como reutilizables (y opcionalmente adjuntos), sin reutilizar historial/auditoría.

7.13 REQUISITOS NO FUNCIONALES ESPECÍFICOS DEL MOTOR

- **Auditabilidad end-to-end:** eventos inmutables y reconstrucción completa.
- **Seguridad:** acciones autorizadas por RBAC/ABAC, especialmente en excepciones y aprobaciones.
- **Observabilidad:** logs estructurados y correlación de operaciones críticas.

7.14 EJEMPLO DE PLANTILLA PARA TRAMITE DE INICIO

Anexo 10.1.2

8 ENTIDADES Y FLUJOS PRINCIPALES

El presente modelo de entidades constituye una propuesta inicial para la estructura de datos del Sistema de Gestión de Trámites (SITDF). Su propósito es identificar los principales objetos de información, sus relaciones y el alcance funcional que deberá soportar el sistema.

El modelo se organiza en torno a los núcleos operativos del sistema: la gestión de trámites, la administración de productos y proyectos industriales, el control de documentación y certificados, y la trazabilidad de la producción declarada por las empresas.

En el eje central se encuentran las entidades Trámite y Tipo de Trámite, que representan respectivamente las instancias concretas en curso y los modelos o plantillas que definen su estructura, pasos, requisitos y documentos requeridos. La separación entre la plantilla y la instancia permite mantener versiones históricas y adaptar los procesos administrativos sin afectar los trámites ya iniciados.

Las entidades Empresa, Apoderado y Usuario conforman el dominio organizacional y de acceso al sistema. Se complementan con los componentes Rol, Habilidad y las tablas de asignación correspondientes, que determinan las acciones que cada actor puede realizar. Esta estructura permite una gestión granular de permisos y la adaptación a distintos perfiles institucionales y empresariales.

El dominio productivo está representado por las entidades Proyecto y Producto, junto con sus componentes técnicos: Especificaciones, Procesos Productivos, Operaciones y Equipamientos. Estos elementos describen las características técnicas y operativas de los bienes producidos bajo el régimen, y sirven de base para los trámites asociados, como ampliaciones, desafectaciones o autorizaciones especiales.

El subsistema de producción declarada incorpora las entidades Declaración de Producción, Ítems de Producción y Soportes Documentales, destinadas a registrar la información estadística y de seguimiento remitida por las empresas de manera periódica.

La gestión documental se unifica en la entidad Documento, que agrupa actas, informes técnicos, decisiones administrativas y otros registros oficiales. Cada documento puede estar firmado digitalmente mediante la entidad FirmaDigital y vincularse a los Certificados emitidos por la autoridad competente, asociados a un trámite, producto o empresa.

Finalmente, las entidades Notificación y EventoAuditoría garantizan la trazabilidad de las comunicaciones y acciones realizadas dentro del sistema.

Debe señalarse que este modelo tiene carácter preliminar y será ajustado a medida que avance el proyecto y se profundice el relevamiento funcional y técnico. Existen detalles de algunas entidades —como estructuras internas de documentos, validaciones de datos o relaciones específicas entre productos y proyectos— que dependen de definiciones aún en desarrollo. Por tanto, el modelo presentado no debe interpretarse como una definición final ni exhaustiva, sino como una base de trabajo estructural sobre la cual evolucionará el diseño de la arquitectura de datos definitiva del sistema.

8.1 DIAGRAMA ENTIDAD RELACION INICIAL PROPUESTO

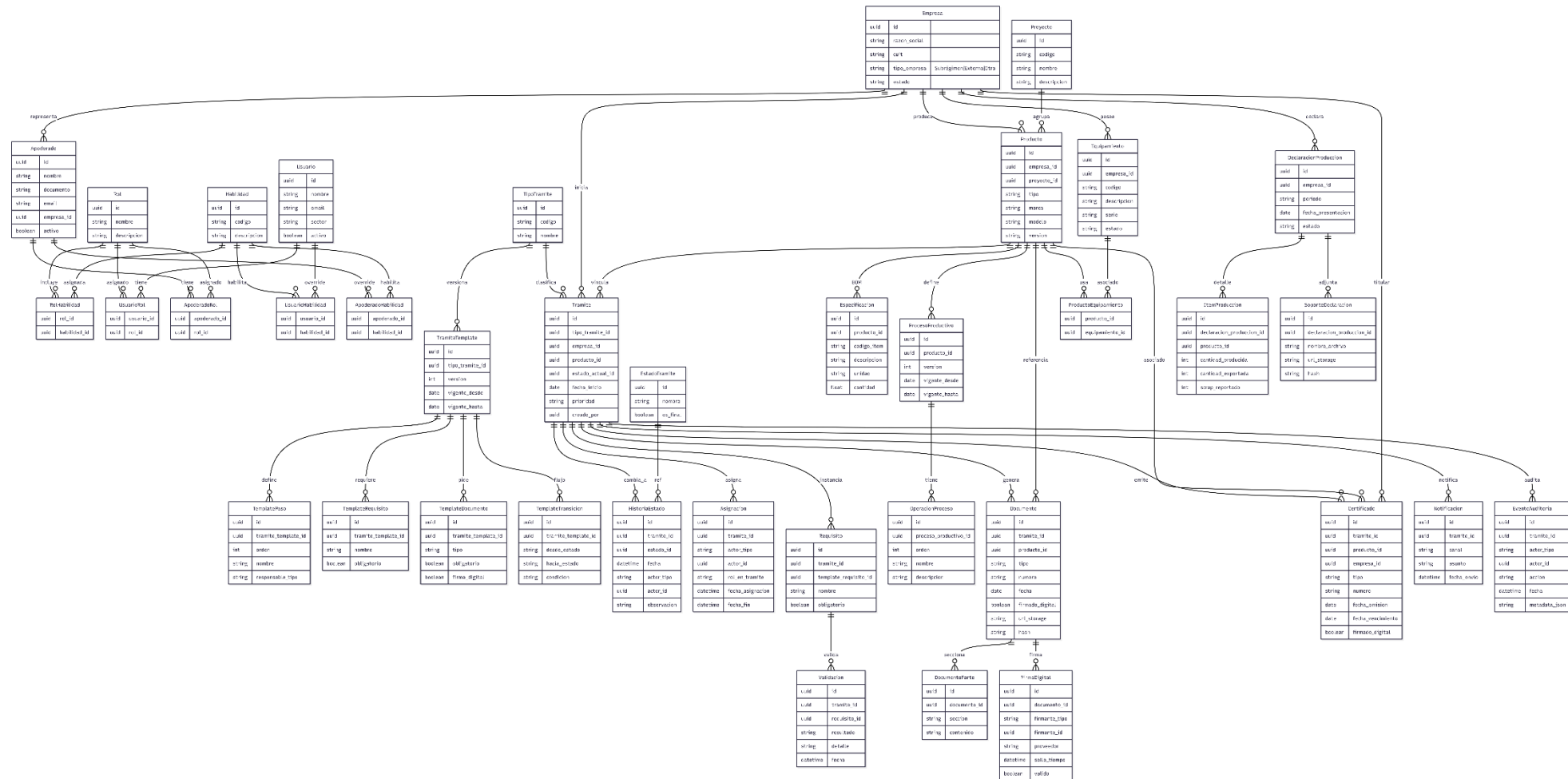


Figura 5: Diagrama preliminar de entidades – relaciones

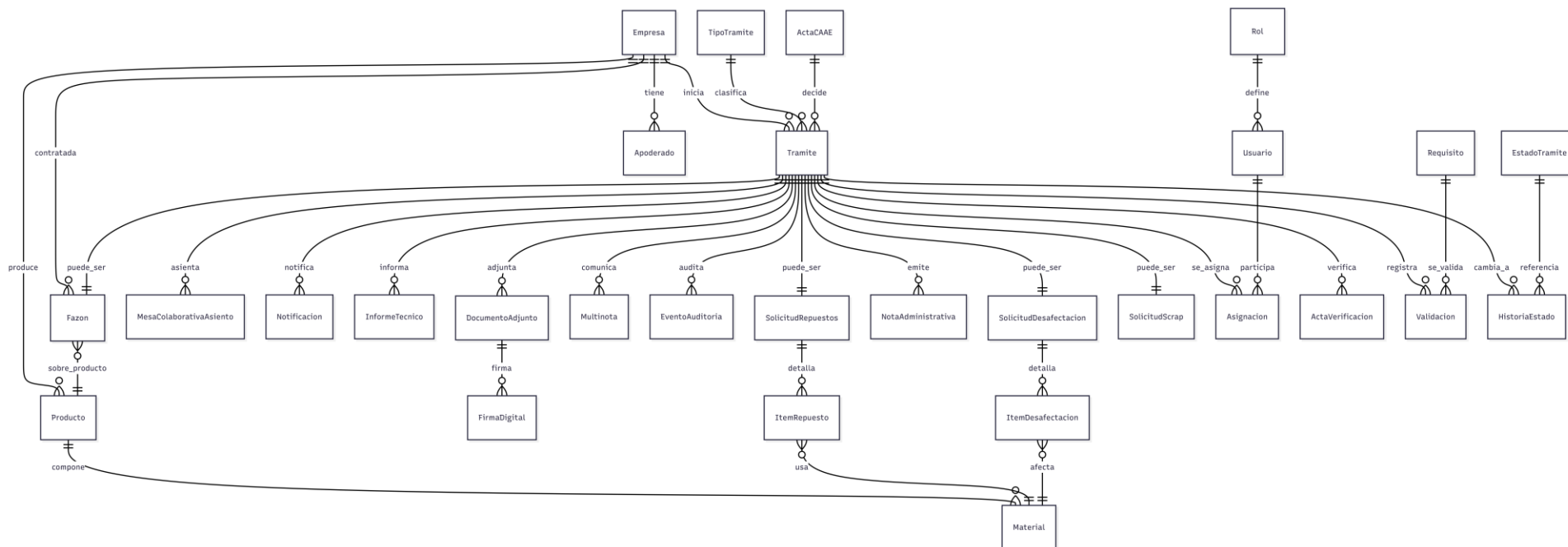


Figura 6: Diagrama preliminar de entidades simplificado con relacion de acciones.

8.2 MODELO DE DATOS MODULO MOTOR DE TRAMITES

En esta sección se describe un **esquema conceptual inicial**. La estructura final podrá evolucionar conforme avance el relevamiento y el desarrollo, manteniendo los principios de versionado, auditabilidad y compatibilidad hacia atrás.

8.2.1 Modelo de Datos: objetivos, alcance y principios

El modelo de datos del SITDF debe soportar:

- i. trámites altamente configurables y versionados,
- ii. trazabilidad/auditoría completa,
- iii. gestión documental con firma digital,
- iv. consultas operativas (tableros) y analíticas (reportes/KPIs), y
- v. integración futura con sistemas externos.

8.2.2 Stack de persistencia y criterios generales

- Base de datos relacional (PostgreSQL) como fuente de verdad transaccional.
- Uso de columnas JSONB para estructuras dinámicas (definiciones de flujo, datos de formularios, metadatos de auditoría).
- Almacenamiento de archivos en object storage (MinIO/S3); en BD solo metadatos, hashes e índices.
- Cache/cola liviana (Redis) para optimizar lecturas repetidas (p. ej., tableros y lookups).
- Convención de identificadores UUID, timestamps (created_at/updated_at) y borrado lógico (deleted_at) en entidades críticas.

9 Núcleos de entidades y dominios

Para facilitar implementación, mantenimiento y performance, el modelo se organiza en dominios (bounded contexts) con relaciones explícitas:

9.1.1.1 Dominio Organizacional y Acceso

- Empresa: entidad principal del sujeto regulado (CUIT único).
- Apoderado: representantes habilitados de la empresa, con capacidad (eventual) de firma digital.
- Usuario: identidad interna/externa con atributos de sector y estado (activo/inactivo).
- Rol: conjunto de permisos (idealmente normalizado en tablas de permisos/habilidades).

9.1.1.2 Dominio de Trámites (núcleo transaccional)

- TipoTramite: define el modelo/versiones del flujo, formularios, reglas, y parámetros del trámite.
- Tramite: instancia concreta; referencia a empresa y a la versión del tipo de trámite.
- EstadoTramite: catálogo de estados globales y/o técnicos.
- HistoriaEstado: registro inmutable de transiciones de estado (quién/cuándo/observación).
- Asignacion: participación de usuarios/sectores en una instancia (responsable, revisor, inspector, etc.).

9.1.1.3 Dominio de Validación y Requisitos

- Requisito: definición versionada de reglas/controles aplicables por tipo de trámite.

- Validación: evidencia de ejecución de un requisito sobre un trámite (resultado, detalle, fecha).

9.1.1.4 Dominio Documental y Comunicaciones

- DocumentoAdjunto: metadatos de archivo asociado al trámite (almacenado en MinIO/S3).
- FirmaDigital: registro de firmas sobre documentos (proveedor, sello de tiempo, validez).
- Notificacion: registro de comunicaciones emitidas (in-app/correo) y su trazabilidad.
- Multinota / NotaAdministrativa / ActaVerificacion / InformeTecnico / ActaCAAE: documentos “tipificados” que pueden unificarse o mantenerse como tablas específicas según estrategia (ver punto 9.1.4).
- MesaColaborativaAsiento: comentarios/asientos cronológicos de trabajo colaborativo.
- EventoAuditoria: registro append-only de acciones del sistema (manuales y automáticas).

9.1.1.5 Dominio Productivo y Materiales

- Producto: entidad del catálogo de la empresa (tipo, marca, modelo, etc.).
- Material: componentes o materiales asociados al producto (código, descripción).

9.1.1.6 Subdominios por tipo de trámite (entidades específicas)

Algunos tipos de trámite requieren entidades específicas (solicitudes y sus ítems). El esquema propuesto contempla tablas dedicadas para casos con alta estructura y necesidad de reporting:

- SolicitudRepuestos + ItemRepuesto (referencia a Material).
- SolicitudDesafectacion + ItemDesafectacion (referencia a Material).
- SolicitudScrap.
- Fazon (relación con empresa contratada y producto).

9.1.2 Versionado explícito de TipoTramite (plantilla de flujo)

Para garantizar compatibilidad hacia atrás y trazabilidad, se recomienda modelar explícitamente el versionado del TipoTramite. Esto puede implementarse como una entidad adicional TipoTramiteVersion (o equivalente).

- TipoTramite (estable): código, nombre, descripción, estado (activo/inactivo).
- TipoTramiteVersion (mutable y versionada): definition_jsonb del flujo (nodos, edges, reglas), form_schemas_jsonb, parámetros, fecha_publicación, estado (borrador/publicado/deprecado).
- Tramite debe referenciar TipoTramiteVersion (no solo TipoTramite) para fijar el contrato de ejecución.

9.1.3 Estructuras dinámicas: formularios, reglas y datos de instancia

El motor de trámites requiere persistir datos altamente variables por tipo y versión. Se recomienda un enfoque híbrido: JSONB para payload completo + materializaciones selectivas para consulta/analítica.

9.1.3.1 Payload JSONB de instancia

- Tramite.instance_data_jsonb: respuestas de formularios, por nodo/paso (clave = nodeId), incluyendo adjuntos lógicos y metadatos de validación.
- Tramite.instance_state_jsonb: estado técnico (node actual, flags, excepciones vigentes, checkpoints).

9.1.3.2 Índice EAV o materialización de atributos “hot”

Para soportar búsquedas y reportes sin degradar performance por consultas complejas sobre JSON, se recomienda un índice secundario de atributos clave. Opciones:

- Tabla EAV: TramiteAtributo(tramite_id, key_path, value_text / value_num / value_date / value_bool, updated_at).
- Columnas denormalizadas en Trámite para campos de consulta frecuente (p. ej., expediente_numero, producto_id, sector_actual, responsable_id).
- Vistas materializadas (refresh programado) para métricas (duración por estado, volumetría por período).

9.1.3.3 Ejecución de nodos y tareas

Para reconstrucción precisa de la secuencia de eventos (y no solo estados globales), se recomienda persistir un log de ejecución por nodo:

- TramiteNodoEjecucion(tramite_id, node_id, tipo, status, started_at, completed_at, actor_id, output_jsonb, excepcion_jsonb).
- TramiteTarea(tramite_id, node_id, assigned_to_user_id/role/sector, status, created_at, completed_at, payload_jsonb).

9.1.4 Estrategia documental: unificación vs tablas tipificadas

El diagrama inicial contiene múltiples tablas por tipo documental. Se proponen dos estrategias compatibles:

Estrategia A — Documento unificado (recomendado para escalabilidad):

- Documento(id, tramite_id, tipo, numero, fecha, estado, payload_jsonb).
- Subtipos como valores enumerados (tipo = ACTA_VERIFICACION, INFORME_TECNICO, NOTA_ADMIN, etc.).
- Tablas específicas solo cuando existan campos críticos para reporte masivo o integridad (ítems, cantidades).

Estrategia B — Tablas tipificadas (recomendado cuando los campos/queries son estables y críticos):

- Mantener tablas NotaAdministrativa / ActaVerificacion / InformeTecnico / Multinota / ActaCAAE.
- Agregar una entidad DocumentoBase para unificar metadatos y firma (FK desde tablas específicas).

9.1.5 Índices principales (recomendación inicial)

Los siguientes índices priorizan tableros, navegación de trámites y auditoría. Nombres ilustrativos; ajustar a motor SQL seleccionado.

9.1.5.1 Empresa / Usuario / Rol

- Empresa: UNIQUE(cuit); INDEX(razon_social) con soporte de búsqueda (p. ej., trigram); INDEX(estado).
- Usuario: UNIQUE(email); INDEX(rol_id); INDEX(sector, activo).
- Rol: UNIQUE(nombre).

9.1.5.2 *Trámite y ejecución*

- Trámite: INDEX(empresa_id, fecha_inicio DESC); INDEX(estado_actual_id); INDEX(tipo_trámite_version_id);
- Trámite: INDEX(responsable_id, estado_actual_id); INDEX(updated_at DESC) para tableros.
- HistoriaEstado: INDEX(trámite_id, fecha DESC); INDEX(estado_id, fecha DESC) para métricas.
- Asignación: INDEX(usuario_id, fecha_fin) + filtro por fecha_fin IS NULL (asignaciones activas); INDEX(trámite_id).
- TrámiteNodoEjecución: INDEX(trámite_id, started_at DESC); INDEX(node_id, completed_at DESC).

9.1.5.3 *Validaciones, documentos, auditoría*

- Validación: INDEX(trámite_id, fecha DESC); INDEX(requisito_id); INDEX(resultado).
- DocumentoAdjunto: INDEX(trámite_id); UNIQUE(hash) opcional (deduplicación); INDEX(tipo).
- FirmaDigital: INDEX(documento_id); INDEX(firmante_id, sello_tiempo DESC).
- Notificación: INDEX(trámite_id, fecha_envío DESC); INDEX(canal).
- EventoAuditoría: INDEX(trámite_id, fecha DESC); INDEX(actor_id, fecha DESC); GIN(metadata_jsonb) para búsquedas puntuales por claves.

9.1.6 Consultas operativas y analíticas prioritarias

9.1.6.1 *Tablero de usuario (Secretaría de Industria)*

- Trámites donde el usuario es responsable máximo o tiene asignación activa.
- Filtros por estado global, sector, tipo de trámite, prioridad, y vencimientos.
- Orden: prioridad DESC, updated_at DESC (o fecha_fin estimada).

9.1.6.2 *Tablero de empresa*

- Trámites por empresa_id; filtros por tipo, estado, período.
- Acción de clonado: copiar subset de instance_data_jsonb “reutilizable” (según plantilla).

9.1.6.3 *Vista detalle de trámite*

- Datos maestros (empresa/producto), estado actual, tareas pendientes, formularios completados (JSONB), documentos y firmas.
- Timeline unificado: HistoriaEstado + TrámiteNodoEjecución + EventoAuditoría + MesaColaborativaAsiento.

9.1.6.4 *Reporting/KPIs*

- Volumetría por tipo/estado/período (series temporales).
- Duración media/percentiles por estado y por nodo (bottlenecks).
- Tasa de excepciones (nodos exceptuados) y causas frecuentes.

9.1.7 Integridad, consistencia y concurrencia

- FKs con ON DELETE RESTRICT para entidades maestras; ON DELETE CASCADE solo en hijas estrictas del trámite (ítems, adjuntos, eventos) si aplica.
- Idempotencia en nodos de procesamiento (idempotency_key por acción) para reintentos seguros.

9.1.8 Retención, archivado y borrado lógico

- Retención mínima de auditoría y eventos conforme política definida en documento.
- Borrado lógico (deleted_at) para entidades sensibles, manteniendo trazabilidad.
- Archivado de trámites finalizados: partición/compresión por período para tablas de alto volumen (EventoAuditoria, HistoriaEstado, NodoEjecucion).

9.1.9 Recomendaciones específicas para búsqueda

- Búsqueda por identificadores: id, número de documento, CUIT, expediente.
- Búsqueda textual: razon_social, asunto de notificación, notas/asientos (full-text y/o trigram).
- Búsqueda en JSONB: limitar a claves acordadas (hot keys) con GIN parcial o materialización EAV.

9.1.10 Diccionario de datos (entidades principales)

A continuación, se describe el propósito y recomendaciones por entidad (no exhaustivo).

9.1.10.1 Empresa

- Propósito: sujeto regulado; núcleo de acceso y titularidad de trámites.
- Claves/constraints: PK UUID; CUIT UNIQUE; estado (catálogo) para habilitaciones.
- Índices: UNIQUE(cuit); búsqueda por razón social (trigram/full-text).
- Notas: considerar tabla EmpresaDomicilio para múltiples domicilios.

9.1.10.2 Apoderado

- Propósito: representantes de empresa; canal de firma y notificaciones.
- Claves/constraints: FK empresa_id; email UNIQUE por empresa (opcional).
- Índices: (empresa_id); (email).
- Notas: modelar vigencia/mandato y documentos habilitantes (adjuntos) en DocumentoAdjunto/Documento.

9.1.10.3 Usuario / Rol

- Propósito: identidades internas/externas y control de acceso.
- Constraints: email UNIQUE; rol_id FK; activo boolean.
- Índices: (rol_id), (sector, activo).
- Recomendación: separar permisos en tablas normalizadas (Permiso, RolPermiso) para evitar JSON opaco.

9.1.10.4 TipoTramite / TipoTramiteVersion

- Propósito: catálogo del trámite y su definición versionada (flujo, formularios, reglas).
- Constraints: TipoTramite.codigo UNIQUE; TipoTramiteVersion.version UNIQUE por tipo_tramite_id; status enum.
- Índices: (tipo_tramite_id, version DESC); (status).
- Notas: almacenar definition_jsonb y form_schemas_jsonb; validar con esquemas (JSON Schema) en capa de aplicación.

9.1.10.5 Tramite

- Propósito: instancia de ejecución; referencia inmutable a la versión del tipo de trámite.
- Constraints: FK empresa_id; FK tipo_tramite_version_id; estado_actual_id; responsable_id (usuario interno).

- Índices: (empresa_id, fecha_inicio DESC), (estado_actual_id), (responsable_id, estado_actual_id), (updated_at DESC).
- Campos sugeridos: instance_data_jsonb, instance_state_jsonb, node_actual_id, sector_actual, prioridad, created_by.

9.1.10.6 HistoriaEstado

- Propósito: timeline oficial de cambios de estado del trámite (append-only).
- Constraints: FK tramite_id, estado_id, actor_id; fecha NOT NULL.
- Índices: (tramite_id, fecha DESC), (estado_id, fecha DESC).
- Notas: evitar UPDATE/DELETE; corregir con eventos compensatorios.

9.1.10.7 Asignacion

- Propósito: asignaciones operativas por rol/usuario/sector; habilita worklists.
- Constraints: FK tramite_id, usuario_id; fecha_fin NULL = asignación activa.
- Índices: (usuario_id, fecha_fin), (tramite_id).
- Recomendación: permitir asignación por rol/sector además de usuario (para colas); usar columnas nullable o tabla adicional.

9.1.10.8 Requisito / Validacion

- Propósito: reglas verificables y evidencia de su resultado en un trámite.
- Constraints: Requisito versionado por tipo de trámite; Validacion referencia a trámite y requisito.
- Índices: Validacion(tramite_id, fecha DESC), (requisito_id), (resultado).
- Notas: Validacion.detalle puede ser JSONB para estructura; mantener actor_id si es validación manual.

9.1.10.9 DocumentoAdjunto / FirmaDigital

- Propósito: metadatos de archivos y su firma; el archivo vive en MinIO/S3.
- Constraints: DocumentoAdjunto.hash opcional UNIQUE; url_storage inmutable; FirmaDigital vincula documento_id y firmante_id.
- Índices: DocumentoAdjunto(tramite_id), FirmaDigital(documento_id).
- Recomendación: almacenar size_bytes, mime_type, classification, version; y sello de tiempo para firma.

9.1.10.10 Notificacion

- Propósito: trazabilidad de comunicaciones; base para reintentos y auditoría.
- Índices: (tramite_id, fecha_envio DESC), (canal).
- Notas: agregar status (enviada/fallida/reintentando) y destinatario.

9.1.10.11 EventoAuditoria

- Propósito: auditoría transversal de acciones manuales y automáticas (append-only).
- Índices: (tramite_id, fecha DESC), (actor_id, fecha DESC), GIN(metadata_jsonb).
- Notas: usar metadata_jsonb para atributos variables (node_id, ip, user_agent, cambios, etc.).

9.1.11 Patrones de consulta (consultas principales)

Se listan consultas típicas que deben guiar índices y materializaciones. *Pseudocódigo SQL a modo de ejemplo:*

9.1.11.1 Tablero de usuario (trámites asignados / responsabilidad)

```
SELECT t.id, tt.codigo, tt.nombre, t.prioridad, t.updated_at,  
       e.razon_social, est.nombre AS estado,  
       t.node_actual_id, t.sector_actual  
FROM tramite t  
JOIN empresa e ON e.id = t.empresa_id  
JOIN tipo_tramite_version ttv ON ttv.id = t.tipo_tramite_version_id  
JOIN tipo_tramite tt ON tt.id = ttv.tipo_tramite_id  
JOIN estado_tramite est ON est.id = t.estado_actual_id  
LEFT JOIN asignacion a ON a.tramite_id = t.id AND a.fecha_fin IS NULL  
WHERE (t.responsable_id = :user_id OR a.usuario_id = :user_id)  
      AND t.deleted_at IS NULL  
ORDER BY t.prioridad DESC, t.updated_at DESC  
LIMIT :page_size OFFSET :offset;
```

9.1.11.2 Timeline reconstruible (estado + eventos + documentos)

-- Vista/unión lógica (puede implementarse como view):

```
SELECT 'ESTADO' AS tipo, he.fecha AS ts, he.estado_id, he.actor_id, he.observacion,  
NULL::jsonb AS meta  
FROM historia_estado he WHERE he.tramite_id = :tramite_id  
UNION ALL  
SELECT 'AUDITORIA', ea.fecha, NULL, ea.actor_id, ea.accion, ea.metadata_jsonb  
FROM evento_auditoria ea WHERE ea.tramite_id = :tramite_id  
UNION ALL  
SELECT 'DOC', d.created_at, NULL, d.actor_id, d.nombre_archivo, jsonb_build_object('tipo',  
d.tipo, 'url', d.url_storage)  
FROM documento_adjunto d WHERE d.tramite_id = :tramite_id  
ORDER BY ts ASC;
```

9.1.11.3 Búsqueda por empresa/documento/identificadores

-- Ejemplo: búsqueda por CUIT o razón social

```
SELECT e.id, e.cuit, e.razon_social
```

```

FROM empresa e
WHERE e.cuit = :cuit
      OR e.razon_social ILIKE '%' || :q || '%'
ORDER BY e.razon_social
LIMIT 50;

```

9.1.11.4 Reporting: duración por estado

-- Duración por estado usando historia de estados

```

WITH x AS (
    SELECT tramite_id, estado_id, fecha AS ts,
           LEAD(fecha) OVER (PARTITION BY tramite_id ORDER BY fecha) AS ts_next
    FROM historia_estado
    WHERE fecha >= :from AND fecha < :to
)
SELECT estado_id,
       COUNT(*) AS transiciones,
       AVG(EXTRACT(EPOCH FROM (COALESCE(ts_next, NOW()) - ts))) AS
duracion_prom_seg
FROM x
GROUP BY estado_id;

```

9.1.12 Consideraciones de performance y escalabilidad

- Tablas de alto volumen: EventoAuditoria, HistoriaEstado, TramiteNodoEjecucion, Notificacion. Considerar partición por mes (RANGE por fecha) cuando volumetría lo justifique.
- Índices parciales: por ejemplo, Asignacion(fecha_fin IS NULL) para asignaciones activas; Tramite(estado_actual_id IN (...)) para estados frecuentes.
- Materialized views para reportes repetitivos (duraciones, volumetrías) y refresh programado.
- Estrategia de JSONB: limitar queries complejas; extraer hot keys a EAV/denormalización.
- Cache: listas de catálogos (Estados, Tipos de trámite, Roles) y prefetch de datos maestros para evaluaciones condicionales.

9.1.13 Relaciones polimórficas: asociación de Trámite con entidades variables

Dado que según el tipo de trámite pueden asociarse entidades adicionales (Producto, Proceso, Proyecto u otras), se recomienda evitar proliferación de columnas nullable.

Alternativas propuestas:

- Tabla TramiteRelacionEntidad(tramite_id, entidad_tipo, entidad_id, rol) con constraints y/o claves foráneas lógicas.
- Contexto JSONB en Tramite.instance_state_jsonb (solo si no se requieren joins frecuentes).
- Columnas directas solo para entidades core muy consultadas (p. ej., producto_id).

10 DESPLIEGUE

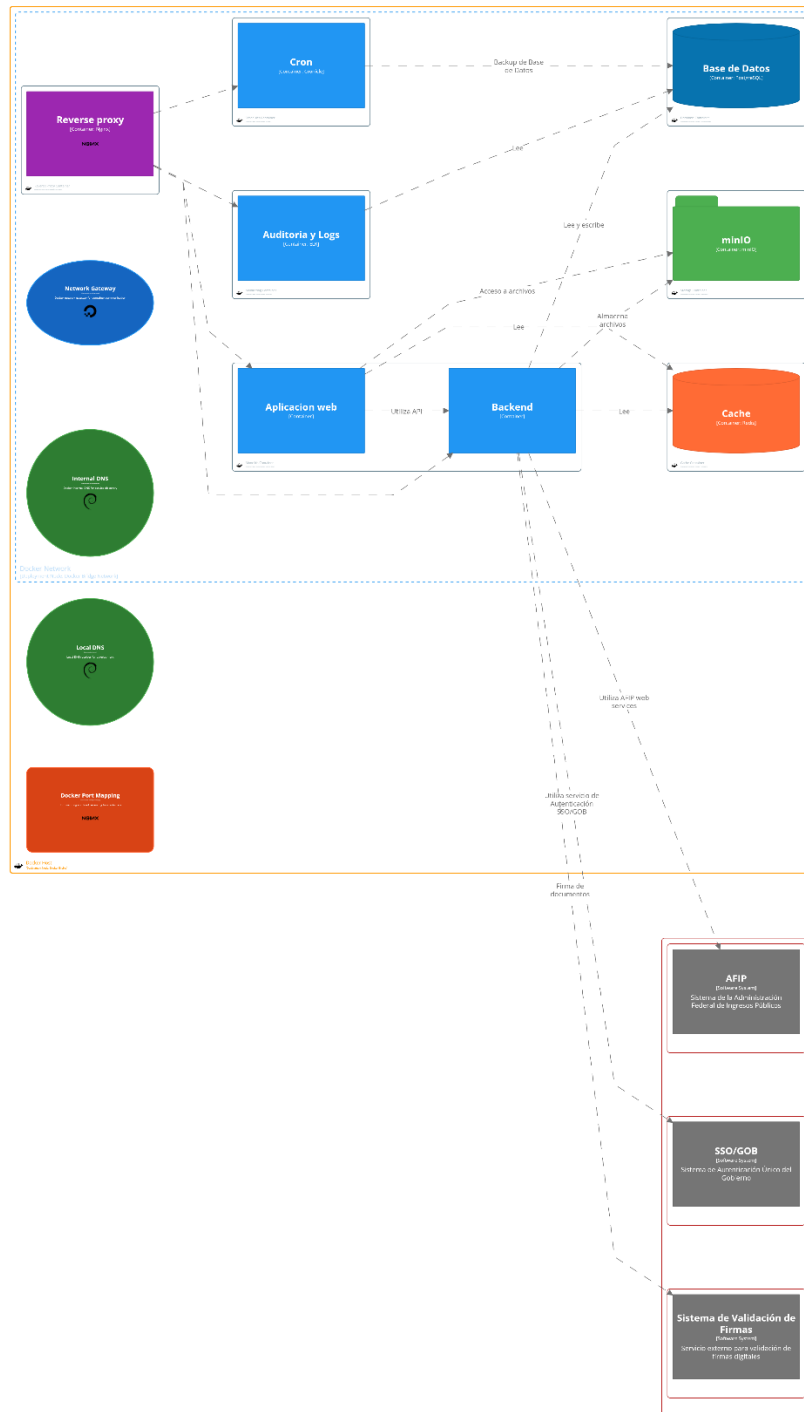


Figura 7: Vista de infraestructura de sistema

El diagrama de despliegue ilustra la arquitectura del Sistema Integral de Gestión de Trámites (SIGT), diseñado para centralizar y automatizar la gestión de trámites de la Secretaría de Industria provincial. Como se mencionó, la solución se basa en una arquitectura modular y en contenedores (Docker), lo que permite su despliegue tanto en entornos *on-premise* como en la nube, garantizando trazabilidad, auditoría, flexibilidad y seguridad.

10.1 INFRAESTRUCTURA DE DESPLIEGUE CON DOCKER COMPOSE (ON-PREMISE)

El Sistema Integral de Gestión de Trámites (SIGT) será desplegado en una única instancia virtual (o servidor físico) *on-premise*, utilizando Docker Compose para orquestar todos los contenedores y sus dependencias. Esta propuesta garantiza un entorno autocontenido, reproducible y fácil de gestionar.

1. Servidor On-Premise e Instancia Virtual:
 - a. Servidor Host: Un servidor físico en el centro de datos local del gobierno provincial. Este servidor debe contar con recursos suficientes (CPU, RAM, almacenamiento SSD de alto rendimiento) para soportar la carga de trabajo de todos los contenedores.
 - b. Instancia Virtual: Sobre el servidor host, se provisionará una instancia virtual (VM) con un sistema operativo Linux (preferentemente Ubuntu Server o Debian) optimizado para contenedores. Esta VM será el entorno donde se instalará Docker Engine y Docker Compose.
 - c. Certificados HTTPS: provistos y administrados por la Agencia de Infraestructura; solicitud vía formulario oficial (pendiente). Claves y certificados almacenados como secretos; renovación programada.
 - d. Dominio: FQDN de acceso sitdf.gobtdf.gov.ar. Límite de un único nivel de subdominio. Redirección HTTP→HTTPS.
2. Arquitectura de Contenedores con Docker Compose: Todos los componentes del SIGT se ejecutarán como contenedores Docker, definidos y orquestados mediante un único archivo docker-compose.yml. Esto incluye:
 - a. nginx (Reverse Proxy/WAF):
 - i. Puerto de Acceso: Expondrá los puertos 80 (HTTP) y 443 (HTTPS) de la VM al exterior. Todas las solicitudes entrantes al sistema pasarán a través de NGINX.
 - ii. Configuración: Manejará la terminación SSL/TLS, reenviando el tráfico cifrado (HTTPS) al backend. Actuará como un *Web Application Firewall* (WAF) ligero, aplicando reglas básicas de seguridad.
 - iii. Red: Se conectará a la red interna de Docker para comunicarse con el contenedor del backend.
 - iv. Certificados TLS provistos por la Agencia; HSTS habilitado; upstreams para SPA y API en el FQDN indicado.
 - b. Aplicación web (Frontend SPA):
 - i. Contenedor que servirá los archivos estáticos de la aplicación frontend (React, Angular, Vue).
 - ii. Acceso: NGINX configurará un upstream para servir directamente estos archivos o proxy a este contenedor si es necesario, aunque en una SPA, NGINX generalmente servirá directamente los assets y las peticiones a la API se proxyarán al backend.

- iii. Volumen: Puede usar un volumen para almacenar los archivos de build de la SPA.
- c. Backend (API REST Monolito Principal):
 - i. Contenedor que aloja la lógica de negocio principal y la API REST.
 - ii. Acceso: NGINX redirigirá las solicitudes a la API a este contenedor. Su puerto interno (ej. 8080) estará expuesto solo a la red interna de Docker.
 - iii. Interacción: Se comunicará internamente con los contenedores de db, minio, cache, y auditoria_logs a través de sus nombres de servicio Docker (DNS interno).
- d. Base de Datos (PostgreSQL/MariaDB/MySQL):
 - i. Volumen: Fundamental. Se montará un volumen persistente Docker para almacenar los datos de la base de datos (/var/lib/postgresql/data o similar). Esto asegura que los datos no se pierdan si el contenedor es recreado o actualizado.
 - ii. Acceso: Solo accesible internamente por el contenedor backend. No expondrá puertos al host de la VM.
 - iii. Recursos: El contenedor backend realizará operaciones de escritura y lectura intensivas sobre este volumen para todas las transacciones de los trámites, gestión de usuarios, roles, historiales, etc.
 - iv. Backups de base de datos: diarios últimos 30 días, primer día de cada mes y copia de fin de año; pruebas de restauración periódicas (ejecutadas desde el contenedor Cron).
- e. minio (Almacenamiento de Archivos S3):
 - i. Volumen: Se configurará un volumen persistente Docker para almacenar todos los archivos adjuntos a los trámites, documentos firmados, plantillas, etc. (ej. /data).
 - ii. Acceso: Solo accesible internamente por el contenedor backend. No expondrá puertos al host de la VM, excepto quizás para un acceso de administración puntual si fuera necesario, pero no para la operación normal.
 - iii. Recursos: El contenedor backend realizará operaciones de escritura (subida de documentos) y lectura (descarga de documentos) sobre este volumen a través de la API de MinIO.
- f. Cache (Redis):
 - i. Volumen (Opcional pero recomendado): Si se necesita persistencia de la caché para escenarios de reinicio rápido (RDB snapshotting), se puede montar un volumen para los datos de Redis. Para una caché volátil pura, no es estrictamente necesario un volumen persistente.
 - ii. Acceso: Solo accesible internamente por el contenedor backend. No expondrá puertos al host.
 - iii. Recursos: El contenedor backend realizará operaciones de escritura (almacenamiento de datos temporales, sesiones) y lectura (recuperación de datos cacheados) de alta velocidad.
- g. Auditoria y logs (Elasticsearch/Logstash/Kibana - ELK Stack):
 - i. Volúmenes: Contenedores separados para Elasticsearch, Logstash y Kibana. Elasticsearch requerirá un volumen persistente para almacenar los índices de logs (ej. /usr/share/elasticsearch/data).
 - ii. Acceso: Solo accesible internamente por el backend (para Logstash) y posiblemente un puerto interno expuesto para Kibana (para acceso administrativo y visualización de logs).

- iii. Recursos: El contenedor backend y otros servicios (si generan logs estructurados) realizarán operaciones de escritura continuas para el registro de eventos y auditoría. Kibana realizará operaciones de lectura para la visualización.
 - iv. Formato de logs: JSON con correlationId; directorio/endpoint de exportación hacia la solución interna de la Agencia (además de ELK local).
 - h. Cron:
 - i. Contenedor para la ejecución de tareas programadas (ej. backups de DB, limpieza de logs, generación de reportes programados).
 - ii. Interacción: Podría interactuar con db y minio para tareas de respaldo.
- 3. Servicios externos:
 - a. AFIP y otros servicios externos: Estos son servicios externos que la backend consumirá a través de llamadas HTTP/HTTPS. No son parte del stack de Docker Compose en la instancia *on-premise*, pero su conectividad debe estar garantizada desde la VM.
 - b. Sistema de Validación/Firma: conectividad HTTPS saliente desde la VM; timeouts y reintentos configurables; variables de entorno para URL.
 - c. SSO/GOB (OAuth2/OIDC): integración con el proveedor de la Agencia para usuarios internos; no se despliega en Docker Compose (servicio externo).
- 4. Red de Contenedores (Docker Network):
 - a. Red Única Privada: Todos los contenedores definidos en el docker-compose.yml residirán por defecto en una única red de puente (bridge network) interna y privada, creada automáticamente por Docker Compose (ej. sigt_default).
 - b. DNS Interno: Dentro de esta red, Docker proporciona un servicio DNS interno. Esto significa que los contenedores pueden referenciarse entre sí utilizando sus nombres de servicio definidos en docker-compose.yml (ej., backend se conectará a db, minio, cache). No se requiere configurar un DNS externo o IPs fijas para la comunicación interna.
 - c. Aislamiento: Esta red es completamente aislada del resto de la red del host, lo que mejora la seguridad. Solo NGINX expondrá puertos al exterior.
- 5. Volúmenes Persistentes (Docker Volumes): Los volúmenes Docker son cruciales para la persistencia de datos y el mantenimiento del estado de la aplicación. Se crearán volúmenes explícitos para los siguientes componentes:
 - a. sigt_db_data (para la Base de Datos): Contendrá todos los archivos de datos de PostgreSQL/MariaDB.
 - b. sigt_minio_data (para MinIO): Almacenará todos los documentos y archivos adjuntos del sistema.
 - c. sigt_elk_data (para Elasticsearch): Contendrá los índices de logs y auditoría de Elasticsearch.
 - d. sigt_cache_data (Opcional, para Redis): Si se requiere persistencia de la caché de Redis.
 - e. sigt_agencia_logs_export (para colector institucional): volumen/ ruta compartida para exportar logs hacia la solución interna de la Agencia. Permisos y propiedad acordes al colector.

La definición de estos volúmenes en el archivo docker-compose.yml asegurará su creación y persistencia fuera del ciclo de vida de los contenedores. Si un contenedor muere o se actualiza, los datos almacenados en estos volúmenes permanecerán intactos.

- 6. Puertos de Acceso a través de NGINX:

- a. 80 (HTTP): Redirigido a 443 (HTTPS) por NGINX (por ejemplo, con una redirección 301). Esto asegura que todo el tráfico sea cifrado.
 - b. 443 (HTTPS): Este es el puerto principal de acceso al SIGT desde el exterior. NGINX será el encargado de la terminación SSL/TLS (con certificados Let's Encrypt o propios) y reenviará el tráfico descriptado a los contenedores internos (principalmente aplicacion_web para assets y backend para la API).
 - c. No se exponen otros puertos: La seguridad se maximiza al no exponer directamente los puertos de la base de datos, MinIO, caché o el backend a la red pública o al host, ya que solo NGINX actúa como *gateway*.
7. Flujo de Lectura/Escritura de Recursos por Contenedores:
- a. backend ↔ db:
 - i. Lectura: El backend consulta la base de datos para recuperar información de trámites, usuarios, configuraciones, historiales, roles, etc.
 - ii. Escritura: El backend inserta nuevos registros de trámites, actualiza estados, guarda datos de usuarios, registra auditorías internas, etc.
 - b. backend ↔ minio:
 - i. Escritura: El backend sube documentos adjuntos (PDFs, imágenes), formularios completados, o versiones firmadas digitalmente de documentos al almacenamiento MinIO.
 - ii. Lectura: El backend recupera estos documentos para su visualización en el frontend o para procesarlos (ej., enviarlos al servicio de firma digital).
 - c. backend ↔ cache:
 - i. Escritura: El backend almacena datos de sesión, resultados de consultas frecuentes, o estados temporales para mejorar la velocidad de respuesta.
 - ii. Lectura: El backend recupera estos datos cacheados para evitar consultas repetitivas a la base de datos o cálculos complejos.
 - d. backend → auditoria y logs:
 - i. Escritura (solo): El backend envía eventos de logs y datos de auditoría (quién, cuándo, qué operación, cambios de estado, etc.) a Logstash/Elasticsearch para su almacenamiento inmutable y posterior análisis. Los componentes ELK se encargarán de la persistencia en su volumen.
8. Infraestructura como Código (IaC):
- a. Provisionamiento reproducible con Terraform (VM/red/volúmenes) y Ansible (paquetes/NGINX/compose).
 - b. Sin configuraciones manuales en servidores; cambios versionados en GitLab.
9. CI/CD (GitLab):
- a. Pipeline build → test → security → package → deploy (entornos dev/test/prod).
 - b. Aprobación manual para prod; artefactos versionados (SemVer) y tags.
10. Gestión de secretos:
- a. Variables protegidas en GitLab y/o volúmenes de secretos; prohibido hardcode en imágenes o repositorio.
 - b. Rotación conforme política de la Agencia.
11. Observabilidad y métricas:
- a. Healthchecks y endpoints /metrics.
 - b. Exportación de logs JSON a ELK y al colector de la Agencia.
 - c. Alertas por errores/timeout en cualquier servicio o integración (firma, SSO) y base de datos.

11 DECISIONES DE ARQUITECTURA

Para la capa de infraestructura y plataforma se ha decidido incorporar únicamente tecnologías maduras y de amplio respaldo comunitario y empresarial. En base de datos, se propone por PostgreSQL y/o MariaDB, por su solidez, extensibilidad y facilidad para desplegar réplicas en clúster, con capacidades nativas de auditoría de transacciones (p.ej. pgAudit). En el nivel de contenedores y orquestación, Docker y Docker Compose o Kubernetes/Docker Swarm permiten empaquetar servicios de forma reproducible y distribuirlos tanto on-premise como en entornos cloud (AWS EKS, Google GKE o Azure AKS), sin imponer dependencia de un proveedor concreto. Para el registro y visualización de logs y métricas se proponen herramientas como ELK Stack (Elasticsearch, Logstash y Kibana) o, alternativamente, la suite Prometheus + Grafana, garantizando que todos los eventos queden almacenados en formatos abiertos (JSON, Prometheus metrics exposition format) para su análisis y exportación.

En el plano de desarrollo de software, la elección de marcos y librerías seguirá criterios de transparencia y simplicidad: Node.js o Spring Boot para servicios backend, React o Vue.js para frontend, siempre con APIs RESTful o RPC documentadas mediante OpenAPI/Swagger y JSON Schema. Esto nos asegura contratos claros entre componentes y facilita la generación automática de clientes y pruebas. El control de versiones, tanto de código como de configuración (Infraestructure as Code), se basará en Git y Terraform/Ansible (si la infraestructura será en la nube), con pipelines CI/CD en GitLab CI, GitHub Actions o Jenkins, según la madurez y familiaridad del equipo. Cada cambio en infraestructura o aplicación se versiona, revisa y prueba en entornos de staging antes de su despliegue en producción.

Para la capa de seguridad y auditoría, implementaremos OAuth2/OIDC para gestión de identidades federadas (si aplicasen) y roles basados en atributos (RBAC), con logs de acceso y autorización almacenados de forma inmutable (append-only). Además, se emplearán proxies inversos como NGINX o Traefik, configurados como WAF (Web Application Firewall) ligero y punto único de entrada SSL/TLS, con certificados gestionados por Let's Encrypt y renovación automática. Las configuraciones de red y cortafuegos se describirán como código (p. ej. en YAML para Kubernetes NetworkPolicies o Terraform para reglas de firewall en la nube), garantizando reproducibilidad y evitando configuraciones “ad hoc” difíciles de auditar.

Por último, la exigencia de mantener un consumo moderado de recursos de hardware se traduce en la adopción de arquitecturas de servicios ligeros, basados en contenedores con imágenes mínimas (Alpine Linux, distroless). Se propone, siempre que sea posible, evitar soluciones pesadas o propietarias que oculten lógica interna (“cajas negras”) y abogando por módulos desacoplados y fáciles de reemplazar. Asimismo, todo el ecosistema se alineará con estándares internacionales y buenas prácticas: ISO/IEC 27001 para gestión de seguridad de la información, OWASP Top 10 para prevención de vulnerabilidades web, y recomendaciones de CNCF para observabilidad y resiliencia. De este modo, aseguramos un sistema fiable, transparente, auditable y sencillo de mantener a lo largo de su ciclo de vida.

11.1 REPOSITORIO Y CONTROL DE VERSIONES (GITLAB)

Para el ciclo de vida de software e infraestructura como código, se adopta Git como sistema de control de versiones y GitLab como plataforma de colaboración, por definición de la Agencia de Innovación. Git es el estándar industrial internacional para versionado distribuido, trazabilidad

fin de cambios y trabajo asincrónico en equipos, soportando ramas, revisiones y auditoría completa del historial. Esta elección asegura contratos claros entre cambios (commits), paquetes liberados (releases) y evidencias de calidad (pipelines), con un flujo reproducible desde desarrollo hasta producción.

En GitLab se mantendrán repositorios oficiales para aplicación, infraestructura (Terraform/Ansible) y documentación (arc42/C4). La estructura del repositorio, los permisos y las convenciones (esquema de branching, nomenclatura y protecciones) serán provistas por la Agencia al inicio del desarrollo; hasta entonces se trabajará con buenas prácticas ampliamente adoptadas:

- Flujo de ramas: Trunk-Based o GitFlow (a confirmar por la Agencia), con ramas `feature/*`, `hotfix/*` y `release/*` y ramas protegidas (`main`, `release/*`).
- Estandarización de commits y versiones: Conventional Commits para semántica de cambios y versionado semántico (SemVer) con tags firmados en cada release.
- Calidad y seguridad integradas: todo cambio ingresa por Merge Request con revisión por pares y pipelines de CI/CD en GitLab (build, test, análisis estático/dinámico, empaquetado y despliegue) antes de llegar a entornos de staging/producción.
- Trazabilidad y auditoría: relación explícita `Issue ↔ MR ↔ Commit ↔ Tag/Release`, templates de Issue/MR, Code Owners en áreas críticas y registro inmutable de evidencias de revisión.
- Gestión de secretos y cumplimiento: variables protegidas y almacenes de secretos; políticas de firma de commits/tags y escaneo de dependencias e imágenes.

Esta decisión alinea el proyecto con prácticas abiertas, reproducibles y auditables, reduce el riesgo operacional y facilita la transferencia a terceros, manteniendo la coherencia con las herramientas institucionales de la Agencia.

11.2 JUSTIFICACIÓN DE LAS DECISIONES TÉCNICAS

11.2.1 Arquitectura modular basada en contenedores (Docker) con orquestación mediante Docker Compose

La arquitectura basada en contenedores ofrece múltiples ventajas y beneficios. En primer lugar, garantiza portabilidad, ya que permite ejecutar el sistema de forma idéntica en distintos entornos —como desarrollo, testing, producción, servidores on-premise o en la nube—, eliminando problemas de configuración o dependencias entre sistemas. Además, facilita el despliegue mediante un solo comando (`docker-compose up`), con el cual se puede levantar toda la infraestructura, incluyendo base de datos, caché, reverse proxy, servicios internos, entre otros, lo que simplifica la puesta en marcha y mejora la reproducibilidad del entorno.

Otra ventaja importante es la separación de responsabilidades: cada módulo se ejecuta en su propio contenedor, lo que favorece el aislamiento, la seguridad, la trazabilidad y el mantenimiento individual sin impactar en el resto del sistema. Aunque no se trata de una arquitectura de microservicios, algunos componentes, como Redis o el reverse proxy, pueden escalarse horizontalmente de manera controlada, sin necesidad de rediseñar toda la arquitectura. Finalmente, Docker Compose genera automáticamente una red virtual interna

donde los servicios se comunican entre sí por nombre, gracias a un sistema de DNS interno, lo que reduce la exposición innecesaria de puertos y simplifica la configuración de red. Las políticas de red, terminación TLS en el reverse proxy y los parámetros de seguridad de cada servicio se versionan como código junto con la definición de `docker-compose.yml`, reforzando reproducibilidad y auditabilidad.

11.2.2 Aplicación principal como monolito

Esta arquitectura presenta varias ventajas y beneficios relevantes. En primer lugar, permite un desarrollo inicial más rápido y cohesivo, ya que mantener la lógica de negocio y los módulos principales —como la API, los flujos, las validaciones y las exportaciones— dentro de un solo servicio facilita el versionado, las pruebas y la coordinación entre equipos. Además, reduce la complejidad operativa al evitar los desafíos típicos de los microservicios, tales como la comunicación entre servicios, la sincronización de datos o la gestión de múltiples versiones, y simplificando la infraestructura requerida considerablemente.

Por otro lado, implica una menor sobrecarga técnica, lo cual resulta ideal para un sistema que se encuentra en una etapa de evolución temprana o media. Este enfoque se adapta especialmente bien a proyectos con requisitos funcionales complejos, pero que aún pueden gestionarse de forma centralizada desde el punto de vista operativo. En línea con la recomendación de la Agencia, se priorizará un monolito sobre tecnologías de amplia adopción interna (p. ej., Python—Django con Django REST Framework en el backend y React en el frontend, base PostgreSQL), sin perjuicio de evaluar alternativas técnicamente justificadas si el proyecto lo requiriera.

11.2.3 Servicios separados para funciones transversales (auditoría, logs, mantenimiento)

Separar los servicios encargados de funciones transversales, como auditoría, logs y mantenimiento, ofrece múltiples ventajas y beneficios. En términos de robustez y mantenibilidad, esta estrategia permite que dichos servicios cuenten con su propio ciclo de vida, mecanismos de escalabilidad y capacidades de monitoreo independientes del sistema principal.

Además, evita sobrecargar al monolito, ya que descarga al servicio principal de tareas intensivas de logging o mantenimiento, lo que mejora su rendimiento y simplifica su lógica interna. Esta separación también permite una mayor especialización tecnológica; por ejemplo, se puede utilizar el stack ELK para la gestión y visualización de logs sin necesidad de integrarla directamente en el código del monolito. La Agencia operará su propia solución interna de observabilidad y logging; el stack ELK del proyecto funcionará en paralelo como complemento para trazabilidad y métricas de negocio. Todos los eventos se emitirán en formato JSON con `correlationId`, y se documentará la integración (`forwarder/directorio`) hacia el colector institucional.

11.2.4 Uso de tecnologías modernas y open source

El uso de tecnologías modernas y open source aporta numerosas ventajas al desarrollo del sistema. En el frontend, el enfoque desacoplado basado en aplicaciones de una sola página (SPA) con tecnologías como React, Vue o Angular permite ofrecer una experiencia de usuario moderna, fluida y fácilmente mantenible.

En el backend, se proponen frameworks priorizando Python—Django (recomendación de la Agencia) y, alternativamente, Express o Spring Boot, todos bien documentados y ampliamente adoptados y ofrecen soporte nativo para middlewares, testing y modularización, lo que facilita el desarrollo escalable y estructurado. A esto se suma el uso de componentes reutilizables y estandarizados, como Redis, MinIO, Keycloak o Metabase, que permiten acelerar el desarrollo mediante soluciones maduras, sostenibles y de comprobada eficacia.

Finalmente, la línea base recomendada por la Agencia es Python — Django — React — PostgreSQL, manteniendo apertura a variantes open source cuando exista justificación técnica registrada (ADR) y sin afectar la interoperabilidad ni el mantenimiento. Es importante también considerar el ecosistema Node.js, ya que brinda beneficios adicionales, como una amplia comunidad, abundante documentación y mayor facilidad para contratar perfiles técnicos, además de favorecer una arquitectura coherente y unificada entre el frontend y el backend.

11.2.5 Autenticación y Autorización centralizada con OAuth2 y OIDC

La autenticación y autorización centralizadas mediante OAuth2 y OpenID Connect (OIDC) representan una decisión clave en la arquitectura de seguridad del sistema. Desde el punto de vista técnico, OAuth2 es un estándar ampliamente adoptado para delegar de forma segura el acceso a recursos protegidos, mientras que OIDC agrega una capa de identidad sobre OAuth2 que permite autenticar usuarios y compartir información de su perfil, facilitando así una gestión segura y estructurada de la identidad.

Esta elección ofrece diversos beneficios. Al tratarse de un estándar interoperable, permite una integración sencilla con sistemas existentes o futuros, como servicios de firma digital, portales de gobierno o servicios externos. Además, habilita el soporte para Single Sign-On (SSO), lo que permite a los usuarios autenticarse una sola vez y acceder a múltiples módulos del sistema, como el frontend, el módulo de exportaciones o la mesa de entradas. También se puede ejercer un control preciso sobre la expiración y revocación de tokens, aplicando políticas estrictas sobre la duración de las sesiones o revocaciones ante comportamientos sospechosos. Asimismo, la arquitectura permite diferenciar flujos de autenticación, como el uso de `authorization_code` para usuarios interactivos o `client_credentials` para procesos internos automatizados.

Usuarios internos (Gobierno): autenticación mediante el SSO de la Agencia (OAuth2/OIDC) ya operativo.

Usuarios externos (Empresas y otros): credenciales gestionadas por el propio sistema, con registro, recuperación y políticas de seguridad (complejidad, expiración, bloqueo por intentos). Se evaluará 2FA (p. ej., TOTP) para fortalecer el acceso externo.

En cuanto a la protección de los contenedores, cada uno de ellos exige un token válido para procesar las peticiones, mientras que el reverse proxy (por ejemplo, NGINX) puede encargarse de validar dichos tokens mediante un middleware o delegar esa validación al backend correspondiente.

11.2.6 Modelo de autorización basado en roles y habilidades (RBAC + ABAC)

El modelo de autorización adoptado combina control de acceso basado en roles (RBAC) con control basado en atributos (ABAC), lo que permite implementar un esquema flexible y granular de permisos. Esta decisión técnica responde a la necesidad de reflejar los distintos niveles de

acceso de los actores del sistema —como empresas, direcciones internas o usuarios externos— en función de sus funciones y responsabilidades específicas.

La combinación de RBAC y ABAC permite definir roles generales y, a su vez, asociar habilidades o capacidades específicas según el contexto. Esto ofrece múltiples beneficios. Por un lado, los roles son configurables y pueden ser administrados de manera dinámica, permitiendo crear perfiles como "Auditor", "director técnico", "Analista de Trámite" o "Empresa", entre otros. Por otro, las habilidades pueden asignarse de forma contextual; por ejemplo, una empresa puede tener permiso para iniciar un trámite, pero no para modificarlo una vez que ha pasado cierta etapa del proceso.

Este enfoque también permite acotar la visibilidad de la información según el contexto. Así, los agentes externos solo pueden acceder a aquellos trámites para los cuales han sido específicamente autorizados, fortaleciendo la seguridad y garantizando el principio de mínimo privilegio.

Nota: la autenticación es diferenciada (SSO para internos; credenciales locales para externos), pero la gestión de roles, permisos y habilidades (RBAC/ABAC) es propia del sistema, centralizada y auditable.

11.2.7 Observabilidad y logs

Los servicios emitirán logs estructurados (JSON) con correlationId y métricas técnicas/funcionales. Se integrará con la solución interna de la Agencia (directorio/forwarder) y se mantendrá ELK del proyecto para análisis y trazabilidad. Se documentarán retenciones, rutas de exportación y tableros/alertas.

11.3 LIMITACIONES

Una de las principales limitaciones de esta arquitectura es la escalabilidad limitada del monolito. Aunque el sistema está contenedorizado, el backend continúa siendo una única aplicación monolítica. Esto implica que, para escalar verticalmente ciertas funcionalidades, debe escalarse toda la aplicación, incluso si solo una parte específica —como las exportaciones o la carga de archivos— es la que requiere mayor capacidad. Si el sistema crece en volumen o complejidad, podría ser necesario desacoplar estas funciones fuera del monolito.

Otra desventaja es el acoplamiento lógico entre módulos internos del monolito. A pesar de que se logra una separación de responsabilidades a nivel de contenedores, la lógica de negocio, los flujos, la autenticación y los procesos asociados a los trámites conviven dentro de un mismo ejecutable. Esta situación puede dificultar la realización de pruebas unitarias, los despliegues independientes o la incorporación de múltiples tecnologías específicas para cada módulo.

También se presenta una limitación en la gestión de errores y fallos, ya que esta se encuentra centralizada. Un error grave en el monolito puede afectar múltiples funcionalidades del sistema, lo que requiere contar con mecanismos sólidos de recuperación y tolerancia a fallos dentro de la misma aplicación para mitigar impactos.

Finalmente, esta arquitectura ofrece una menor granularidad en el monitoreo y el escalado fino. A diferencia de una solución basada en microservicios, en la que cada servicio puede tener métricas, registros y alertas específicas, en este caso gran parte del monitoreo es centralizado

o transversal. Esto puede dificultar diagnósticos ágiles y la aplicación de medidas correctivas puntuales ante problemas específicos.

Adicionalmente, el uso de servicios de firma digital remota introduce dependencias y límites operativos que pueden impactar tiempos de respuesta y tasas de error.

12 GLOSARIO

- **ABAC (Attribute-Based Access Control):** Control de Acceso Basado en Atributos. Un modelo de autorización que regula el acceso a los recursos basándose en los atributos de los usuarios, los recursos y el entorno.
- **ADR (Architecture Decision Record):** Documento corto que registra una decisión técnica clave, su contexto, alternativas consideradas y consecuencias. Facilita trazabilidad y gobierno de arquitectura.
- **AOP (Aspect-Oriented Programming):** Programación Orientada a Aspectos. Un paradigma de programación que busca aumentar la modularidad al permitir la separación de preocupaciones transversales.
- **API (Application Programming Interface):** Interfaz de Programación de Aplicaciones. Un conjunto de definiciones y protocolos que se utiliza para diseñar e integrar software de aplicaciones.
- **AREF:** Agencia de Recaudación del Estado Fuego
- **AFIP - ARCA:** Agencia de Recaudación y Control Aduanero
- **ASVS (Application Security Verification Standard):** Estándar de Verificación de Seguridad de Aplicaciones. Un marco abierto y un estándar de seguridad de aplicaciones.
- **AWS EKS (Amazon Web Services Elastic Kubernetes Service):** Servicio de Kubernetes elástico de Amazon Web Services. Un servicio gestionado de Kubernetes de Amazon.
- **BI (Business Intelligence):** Inteligencia de Negocio. Conjunto de estrategias y herramientas enfocadas en el análisis de datos de una empresa.
- **BPM Engine (Business Process Management Engine):** Motor de Gestión de Procesos de Negocio. Componente encargado de modelar y ejecutar los flujos de los distintos trámites, como máquinas de estados configurables.
- **CAAE:** Comisión del Área Aduanera Especial.
- **CI/CD (Continuous Integration/Continuous Delivery):** Integración Continua/Entrega Continua. Prácticas de desarrollo de software que automatizan la integración de cambios de código y la entrega de software.
- **CNCF (Cloud Native Computing Foundation):** Fundación de Computación Nativa en la Nube. Organización que impulsa tecnologías de código abierto en la nube.
- **CRUD (Create, Read, Update, Delete):** Crear, Leer, Actualizar, Borrar. Las cuatro funciones básicas y esenciales de la persistencia de datos.
- **BD - DB (Database):** Base de Datos. Colección organizada de información estructurada o datos.
- **DNS (Domain Name System):** Sistema de Nombres de Dominio. Sistema que traduce nombres de dominio legibles por humanos en direcciones IP numéricas.

- Docker - Podman: Plataforma de software que permite crear, desplegar y ejecutar aplicaciones dentro de contenedores.
- Docker Compose: Herramienta para definir y ejecutar aplicaciones Docker de múltiples contenedores.
- Docker Swarm: Herramienta de orquestación de contenedores nativa de Docker.
- ELK Stack: Conjunto de herramientas compuesto por Elasticsearch, Logstash y Kibana, utilizado para la gestión de logs.
- Elasticsearch: Motor de búsqueda y análisis distribuido.
- Logstash: Herramienta de procesamiento de datos de código abierto que ingiere datos de múltiples fuentes, los transforma y los envía a un "stash" como Elasticsearch.
- Kibana: Herramienta de visualización de datos de código abierto para Elasticsearch.
- Frontend: La parte de la aplicación con la que el usuario interactúa directamente.
- FQDN (Fully Qualified Domain Name): Nombre de dominio completo del endpoint de una API (p. ej., api.sitdf.gob.ar), usado para DNS, TLS, políticas de CORS y control de tráfico.
- GitLab: Plataforma de forja de software que integra repositorios Git, issues, CI/CD, registro de contenedores y DevSecOps en un solo lugar (self-hosted o cloud).
- GKE (Google Kubernetes Engine): Motor de Kubernetes de Google. Un servicio gestionado de Kubernetes de Google Cloud.
- HTTPS (Hypertext Transfer Protocol Secure): Protocolo Seguro de Transferencia de Hipertexto. Versión segura del HTTP, utiliza SSL/TLS para cifrar las comunicaciones.
- IaC (Infrastructure as Code): Práctica para definir y aprovisionar infraestructura mediante código declarativo o scripts (p. ej., Terraform, Ansible), versionable, reproducible y auditable.
- ISO 9001: Estándar internacional para sistemas de gestión de calidad.
- ISO 25010: Estándar internacional para la calidad del software y los sistemas.
- ISO 27001: Estándar internacional para sistemas de gestión de seguridad de la información.
- JSON (JavaScript Object Notation): Notación de Objetos de JavaScript. Formato ligero de intercambio de datos.
- JWT (JSON Web Token): Token Web JSON. Estándar abierto basado en JSON para crear tokens de acceso que permiten a un usuario verificar su identidad de forma segura.
- KPI (Key Performance Indicator): Indicador Clave de Rendimiento. Métricas que miden el éxito de una actividad o proceso.
- MinIO: Almacenamiento de objetos de código abierto compatible con la API de Amazon S3.
- NGINX: Servidor web y proxy inverso de alto rendimiento.
- Node.js: Entorno de tiempo de ejecución de JavaScript de código abierto y multiplataforma para el desarrollo de aplicaciones del lado del servidor.
- OAuth2 (Open Authorization 2.0): Marco de autorización que permite a una aplicación obtener acceso limitado a una cuenta de usuario en un servicio HTTP.

- **OIDC (OpenID Connect):** Capa de identidad sobre el protocolo OAuth 2.0.
- **OCSP / CRL:** Mecanismos para verificar el estado de un certificado digital: *OCSP* (consulta online en tiempo real) y *CRL* (lista de revocación publicada periódicamente).
- **On-premise:** En las propias instalaciones. Se refiere al software o hardware que se instala y se ejecuta en los servidores y la infraestructura de la organización.
- **OpenAPI/Swagger:** Herramientas para diseñar, construir, documentar y consumir APIs REST.
- **Open Source: Código Abierto.** Software cuyo código fuente está disponible para cualquier persona que desee estudiarlo, modificarlo y distribuirlo.
- **OWASP (Open Web Application Security Project):** Proyecto Abierto de Seguridad de Aplicaciones Web. Comunidad en línea que produce artículos, metodologías, documentación, herramientas y tecnologías de acceso libre.
- **PDF (Portable Document Format):** Formato de Documento Portátil. Formato de archivo universal que conserva las fuentes, imágenes y el diseño original de los documentos.
- **Podman:** Motor de contenedores de código abierto sin demonios para el desarrollo, la gestión y la ejecución de contenedores OCI.
- **PostgreSQL:** Sistema de gestión de bases de datos relacionales de código abierto.
- **Prometheus:** Sistema de monitoreo y alertado de código abierto.
- **Proxy Inverso:** Servidor que se sitúa entre un cliente y uno o más servidores, reenviando las solicitudes del cliente al servidor apropiado.
- **RBAC (Role-Based Access Control):** Control de Acceso Basado en Roles. Un modelo de autorización que restringe el acceso al sistema a los usuarios autorizados en función de su rol.
- **Redis:** Almacén de estructura de datos en memoria de código abierto, utilizado como base de datos, caché y bróker de mensajes.
- **RESTful (Representational State Transfer):** Un estilo arquitectónico para el desarrollo de servicios web.
- **RPC (Remote Procedure Call):** Llamada a Procedimiento Remoto. Un protocolo que permite a un programa de ordenador ejecutar un procedimiento en otro espacio de direcciones (normalmente en otra red) sin que el programador tenga que codificar explícitamente los detalles de esta interacción remota.
- **S3 (Simple Storage Service):** Servicio de Almacenamiento Simple. Servicio de almacenamiento de objetos de Amazon Web Services.
- **SemVer (Semantic Versioning):** Esquema de versionado *MAJOR.MINOR.PATCH*: cambios incompatibles → *MAJOR*, funcionalidades compatibles → *MINOR*, correcciones → *PATCH*.
- **SGC (Sistema de Gestión de Calidad):** Sistema de Gestión de Calidad. (En el contexto del documento, se refiere a un nuevo sistema interno del gobierno provincial).
- **SIGT (Sistema Integral de Gestión de Trámites):** Nombre del prototipo del sistema central para la gestión de trámites.

- SPA (Single Page Application): Aplicación de una Sola Página. Aplicación web o sitio web que cabe en una sola página web con el propósito de dar una experiencia de usuario más fluida, como una aplicación de escritorio.
- SSL/TLS (Secure Sockets Layer/Transport Layer Security): Protocolos criptográficos que proporcionan seguridad en las comunicaciones por internet.
- SSO (Single Sign-On): Inicio de Sesión Único. Un esquema de autenticación que permite a un usuario iniciar sesión con un único ID y contraseña para acceder a múltiples aplicaciones.
- TSA (Time Stamping Authority): Autoridad de sellado de tiempo que emite tokens de tiempo firmados para probar que un documento o firma existía en un instante verificable.
- VM (Virtual Machine): Máquina Virtual. Emulación de un sistema informático.
- WAF (Web Application Firewall): Firewall de Aplicaciones Web. Firewall diseñado para proteger aplicaciones web de ataques.
- YAML (YAML Ain't Markup Language): Un lenguaje de serialización de datos legible por humanos.

12.1 GLOSARIO MODULO MOTOR DE TRAMITES:

- Acción manual (Manual Action Node): Nodo del flujo que requiere intervención humana (operador, empresa, CAAE, director) para completar el paso (revisar, aprobar/rechazar, adjuntar, firmar, etc.).
- Actor: Sujeto (usuario interno/externo) que ejecuta una acción en el sistema y queda registrado en auditoría.
- Asignación (Assignment Node): Nodo que transfiere la responsabilidad/tenencia del trámite o del paso a un rol/usuario/sector.
- Auditoría (Audit Trail): Registro trazable e inmutable de acciones/eventos: quién, qué, cuándo, resultado y evidencia asociada.
- Backward compatibility (Compatibilidad hacia atrás): Capacidad de mantener operativas instancias iniciadas con versiones anteriores de la plantilla sin alterar su historial ni reglas.
- Bypass / Excepción: Mecanismo controlado para saltar un nodo bajo condiciones justificadas y autorizadas, registrando auditoría y efectos sobre reglas dependientes.
- Catálogo de acciones (Acciones preconfiguradas): Conjunto de acciones automáticas estándar reutilizables (p. ej. generar documento, notificar, cambiar estado).
- Condición (Condition Node): Nodo que evalúa automáticamente reglas/expresiones sobre datos del trámite y datos maestros para decidir la transición.
- Correlation ID: Identificador que se propaga en eventos/logs para trazar una operación de punta a punta (observabilidad).
- DTO (Data Transfer Object): Estructura tipada en código usada para transportar y validar datos (reduce el uso de JSON sin control).
- Edge / Transición: Conexión dirigida entre nodos; puede tener condición ("when") que habilita el camino según resultados.

- EAV (Entity–Attribute–Value): Modelo flexible para almacenar atributos dinámicos como pares entidad-atributo-valor, útil cuando el esquema varía por trámite/paso.
- Editor de flujo: Herramienta/interfaz para diseñar nodos/transiciones, configurar reglas y publicar versiones de plantillas.
- Estado global: Estado macro visible del trámite (p. ej. En proceso, En revisión, Pendiente, Finalizado).
- Estado técnico: Estado interno detallado (nodo actual, subestado, responsable, tareas pendientes).
- Evento (Event): Registro atómico de algo ocurrido en la instancia (p. ej. “formulario enviado”, “condición evaluada”, “tarea aprobada”).
- Event store (Registro de eventos): Persistencia append-only de eventos para auditoría y reconstrucción del historial.
- Event sourcing: Patrón donde el estado se deriva (total o parcialmente) de la secuencia de eventos almacenados.
- Excepción: Situación fuera de la regla normal del flujo (p. ej. faltantes, validación externa caída) que requiere tratamiento especial.
- Firma (Signature): Atribución verificable de una acción a un usuario/actor (no repudio interno y/o firma digital según corresponda).
- Formulario (Data Entry / Form Node): Nodo de captura de datos estructurados (campos tipados + adjuntos) que alimenta decisiones y acciones posteriores.
- Grafo (Workflow Graph): Representación del flujo como conjunto de nodos y transiciones dirigidas (inicio único, múltiples finales).
- Handler (Manejador de nodo): Componente que implementa la lógica de ejecución de un tipo de nodo (form/condition/processing/manual/assignment).
- Idempotencia: Propiedad por la cual una acción automática produce el mismo resultado aunque se ejecute más de una vez (clave para reintentos).
- Instancia de trámite: Ejecución concreta de una plantilla para una empresa (y eventualmente producto/proceso), con su propio estado, datos y auditoría.
- Materialización: Estructuras derivadas (tablas/vistas/índices) para consultas rápidas sobre datos originalmente en JSON/EAV.
- Máquina de estados (State Machine): Modelo donde el proceso transita entre estados/nodos según eventos/condiciones.
- Nodo (Node): Unidad de trabajo del flujo (paso). Tipos típicos: start, form, condition, processing, manual, assignment, end.
- Nodo de procesamiento (Processing Node): Nodo que ejecuta una acción automática (persistir, generar documento, notificar, cambiar estado, etc.).
- Observabilidad: Capacidad de entender el comportamiento del sistema mediante logs, métricas y trazas.
- Optimistic locking (Bloqueo optimista): Estrategia para evitar conflictos cuando múltiples actores intentan actualizar el estado del trámite.
- Payload: Datos de negocio asociados a la instancia (p. ej. solicitud, documentos, informes), usualmente almacenados en estructuras flexibles.
- Permisos: Autorizaciones efectivas para ejecutar acciones en un nodo/estado específico (derivadas de roles y/o atributos).

- Plantilla (Template / Tipo de Trámite): Definición versionada del flujo (nodos + transiciones + formularios + reglas + permisos).
- Reintento (Retry): Re-ejecución controlada de nodos automáticos ante fallas; requiere idempotencia y auditoría.
- Regla / Expresión (Rules/Expression): Condición evaluable que decide caminos o valida datos (lenguaje de expresiones configurable).
- Responsable máximo: Usuario interno designado como dueño global del trámite, independientemente de asignaciones por etapa.
- Schema (Esquema): Definición esperada de estructura y tipos de datos (p. ej. JSON Schema para formularios).
- Snapshot: Captura del estado derivado en un punto del tiempo (útil con event store/event sourcing).
- Subflujo: Segmento del proceso que representa una fase (p. ej. post-aprobación, inspección, cierre).
- Tarea (Task): Unidad de trabajo asignada a rol/usuario para resolver un nodo manual; se gestiona en una bandeja.
- Worklist / Bandeja de tareas: Cola/vista donde usuarios ven y gestionan tareas pendientes según rol/sector.
- Versionado (de plantilla): Mecanismo por el cual cada modificación del flujo genera una nueva versión, preservando instancias previas.