📝 Diseño Técnico y Arquitectura

1) 🎯 Alcance y Supuestos

- Automatizar **preparación → validación → publicación → certificación** de versiones.
- Integración con **OneDrive** (almacenamiento) y **Outlook/Teams** (borradores/envío/aviso).
- **Release Notes/Roadmap** automático desde commits/tablero.
- Funcionamiento **local** (offline posible) con adaptador LocalFS y posterior integración cloud.

---

2) ✅ Requisitos

2.1 ⭐ Funcionales (RF)

1. **RF-1. Gestión de versiones y artefactos**
   - **RF-1.1 – Registrar versión**: Registrar una nueva versión con metadatos básicos: número de versión, nombre, fecha de creación y responsable.
   - **RF-1.2 – Almacenar artefactos**: Adjuntar artefactos asociados a una versión (documentos, binarios, notas técnicas).
   - **RF-1.3 – Validar versión**: Validar que metadatos y artefactos cumplen reglas antes de marcar como *lista*.
   - **RF-1.4 – Publicar versión**: Marcar y **notificar** la publicación de una versión aprobada.
2. **RF-2. Notificaciones y aprobaciones**
   - **RF-2.1 – Notificar publicación**: Notificar automáticamente (Outlook/Teams) cuando una versión es aprobada y publicada.
   - **RF-2.2 – Solicitud de aprobación (opcional)**: Dejar borrador y enviar solicitud de validación antes del envío definitivo.
3. **RF-3. Integraciones externas**
   - **RF-3.1 – Repositorios**: Vincular versiones con repositorios de código/artefactos (Git, Nexus, Artifactory).
   - **RF-3.2 – OneDrive**: Almacenar documentos relacionados automáticamente en OneDrive.
   - **RF-3.3 – Outlook**: Disparar notificaciones vía correo/Teams utilizando Outlook/Graph.

2.2 ⭐ No funcionales (RNF)

4. **Rendimiento y escalabilidad**
   - **RNF-1.1 – Tiempo de respuesta**: Operaciones críticas (registro, consulta, publicación) < **2 s** bajo carga normal.
   - **RNF-1.2 – Escalabilidad moderada**: Soportar crecimiento gradual sin migrar de inmediato a microservicios.
5. **Seguridad**
   - **RNF-2.1 – Autenticación y autorización**: Acceso solo a usuarios autenticados, con roles (admin, usuario estándar).
   - **RNF-2.2 – Seguridad en datos sensibles**: Cifrado **en tránsito TLS 1.2+**.
   - **RNF-2.3 – Control de acceso**: Solo usuarios autorizados pueden publicar o aprobar versiones.
6. **Fiabilidad y disponibilidad**
   - **RNF-3.1 – Disponibilidad mínima**: Uptime **99%** horario laboral (L–V, 8–18) *si se ejecuta como servicio*.
   - **RNF-3.2 – Tolerancia a fallos**: Ante fallo de integración (OneDrive/Outlook), registrar error y permitir **reintento manual** sin perder datos.
7. **Usabilidad**
   - **RNF-4.1 – Interfaz intuitiva**: Interfaz web clara (o CLI con ayuda), navegación por **módulos de negocio** (screaming architecture).
   - **RNF-4.2 – Consistencia**: Estilo visual e interacción homogéneos.
8. **Mantenibilidad y extensibilidad**
   - **RNF-5.1 – Arquitectura modular**: Slices independientes que faciliten extensiones/migraciones futuras.
   - **RNF-5.2 – Pruebas automatizadas**: Cobertura ≥ **70%** por slice (unitarias e integración).
9. **Integración**
   - **RNF-6.1 – Estándares de integración**: APIs REST/GraphQL o **SDKs oficiales**.
   - **RNF-6.2 – Logging de integraciones**: Registrar cada evento (Outlook/OneDrive) para diagnóstico.

---

3) 🏛️ Arquitectura

Estilo: Monolito modular con Arquitectura Hexagonal + Vertical Slices + Screaming Architecture.

- **Dominio (core)**: Entidades, Servicios (Crear/Validar/Publicar/Sellar), Reglas/Validadores.
- **Puertos de entrada**: REST (y **CLI opcional**).
- **Puertos de salida**: `Repo` (SQLite), `Storage` (LocalFS/OneDrive), `Notify` (Outbox/Outlook), `ReposRef` (Git/Nexus/Artifactory).
- **Adaptadores**: `SQLiteRepo`, `LocalFSStorage`, `OneDriveStorage`, `LocalOutbox`, `OutlookGraph`.
- **Feature flags (.env)**: `STORAGE=local|onedrive`, `NOTIFY=outbox|graph`.

Justificación: simplicidad operativa local, extensible sin tocar el dominio.

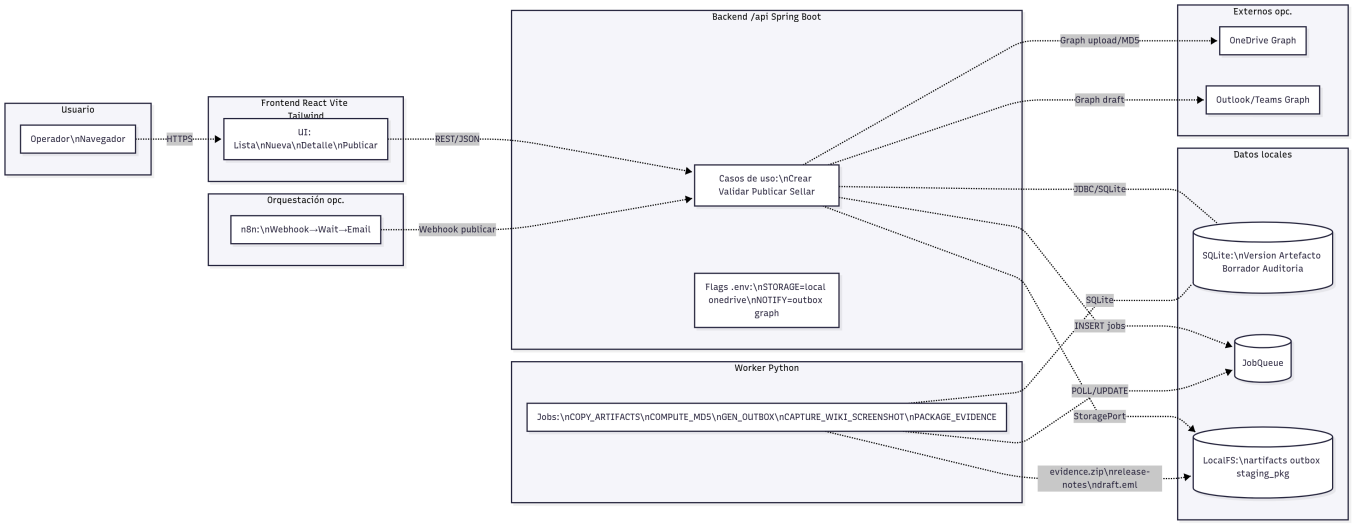
---

4) 📐 Componentes Principales

- **Backend (Spring Boot)**
  - Casos de uso de negocio.
  - Adaptadores:
    - LocalFS (modo offline).
    - OneDrive (almacenamiento).
    - Outlook/Teams (notificaciones).
    - Repositorios (Git/Nexus/Artifactory).
- **Frontend (React opcional)**
  - Formulario para ingresar datos de versión.
  - Dashboard para ver estado de validación/publicación.
  - Vista de release notes generados.
- **Orquestación / Automatización**
  - Opcional con **n8n**, python, **MCP** o agentes IA para ejecutar flujos sin código.
- **Persistencia (opcional)**
  - Base de datos liviana (SQLite, H2, PostgreSQL).
  - O se puede manejar todo con filesystem + logs.

---

5) ☁️ Diagrama de Arquitectura



---

6) 🧰 Stack y herramientas

1. Backend:
   - Lenguaje / Runtime: Java 21 LTS (Temurin / Eclipse Adoptium):
     Base estable en producción, con diseño preparado para migrar a **Java 25 LTS (sep-2025)** sin refactors mayores.
   - Framework: Spring Boot 3.3.5.
   - Dependencias (MAVEN):
     - spring-boot-starter-web:3.3.5`
     - `spring-boot-starter-validation:3.3.5`
     - `spring-data-jdbc:3.3.5`
     - `org.xerial:sqlite-jdbc:3.56.0`
     - `org.flywaydb:flyway-core:9.24.2`
     - `spring-boot-starter-actuator:3.3.5`
     - `org.springdoc:springdoc-openapi-starter-webmvc-ui:2.6.0` (Opcional)
     - `spring-boot-starter-test:3.3.5`
   - IDE: IntelliJ IDEA Community 2024.2+
   - Build Tool: Maven Wrapper 3.9.2
     -
       - Plugins: `maven-enforcer-plugin`, `spotless-maven-plugin`, `owasp-dependency-check`.
2. Orquestación y Automatización:

- **n8n v1.50.7 (Docker latest)**
  - Publicación de versiones, validación y notificación.
  - Integración con OneDrive y Microsoft Graph.
- **MCP / Agentes inteligentes
  - Validaciones inteligentes del release para las observaciones.
  - Generación de Release Notes y borradores de correos con un formato establecido.
  - Integración vía n8n Webhooks/HTTP.
  - Notificaciones para validación humana.
  - Integración vía n8n Webhooks/HTTP (nodo **MCP Client Tool** contra servidor MCP **SSE**).
  - Requisito: **Node.js 20 LTS** para servidores MCP.
  - Gate humano: **Wait for Webhook / Aprobación manual** antes de enviar definitivo.

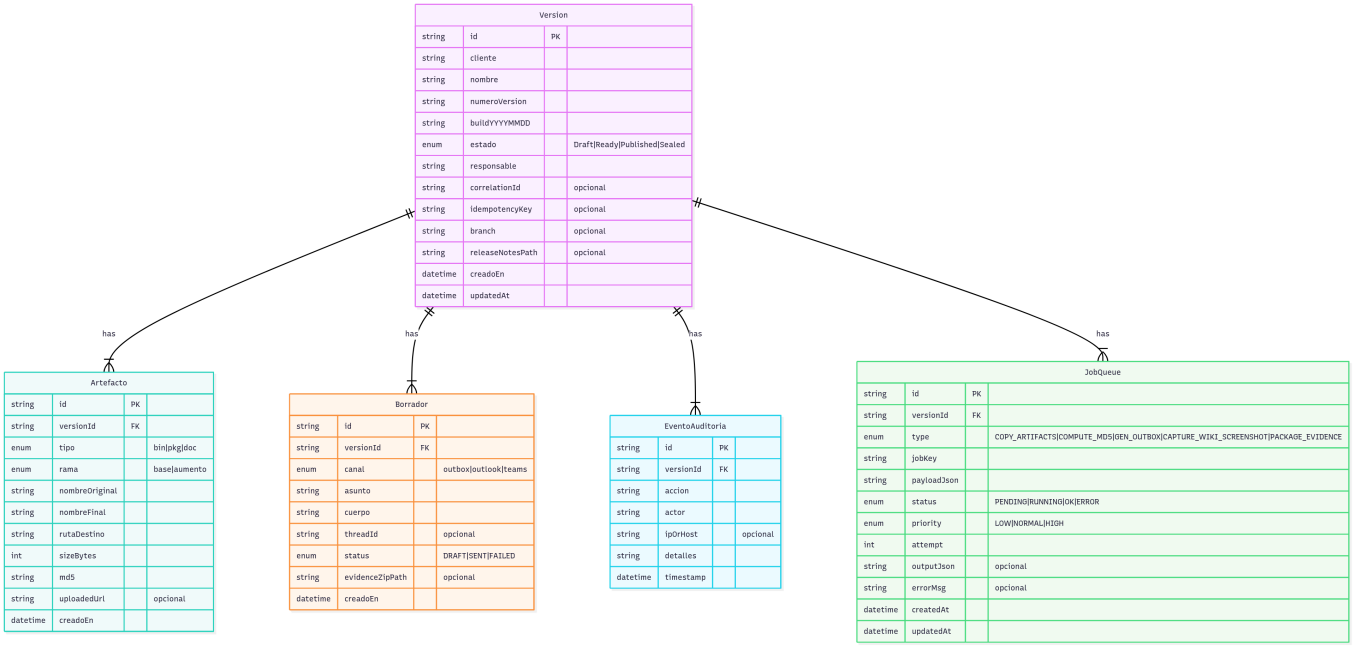
3. Frontend:
   - Framework: React 18.4 con TypeScript 5.3
   - Herramienta de creación y build: Vite v5.0+ (starter react-ts)
   - Dependencias base:
     - react@18.4.0
     - react-dom@18.4.0
     - typescript@5.3.4
   - IDE: VS Code 1.93+
4. Worker (Opcional, última prioridad):
   - **Lenguaje**: Python **3.11.5**
   - **Dependencias**:
     - `jinja2==3.1.2`
     - `playwright==1.44.0` (con `python -m playwright install chromium`)
     - `sqlalchemy==2.0.20` (si se requiere ORM)
     - `hashlib` (stdlib, para checksums)
     - `watchdog==3.1.1` (solo si hace falta monitoreo de archivos; reemplazable con n8n).
   - **IDE**: VS Code (extensión Python) o PyCharm CE 2024.2
5. Base de Datos
   - SQLite con configuraciones de performance y concurrencia:

- journal_mode = WAL
- synchronous = NORMAL
- busy_timeout = 5000
- foreign_keys = ON
- Driver JDBC: sqlite-jdbc:3.56.0
- Migraciones: Flyway 9.24.2

6. Utilidades
   - API Testing: Insomnia 2024.5+
   - Alternativa ligera: Bruno (última release)
   - CLI HTTP: HTTPie 3.7+
   - Contenedores: Docker + Docker Compose
   - CI/CD Pipelines:
     - Plataforma: GitHub Actions / GitLab CI
     - Estrategia:
       - Build & Test en **JDK 21 LTS** y **JDK 25 EA** (dual testing → compatibilidad futura).
       - Stages: *Build* → *Test* → *Package* → *Security Scan* → *Deploy Staging* → *Deploy Prod*.
   - Frontend: CI con Node.js 20, Vite build + Vitest coverage.
   - Orquestación: despliegue de workflows en n8n (Docker).
   - Backend (extra): paso **OWASP dependency-check** en CI.
   - Frontend: `npm run lint` + `vitest run --coverage` en CI.
   - E2E: reintentos **solo** ante 5xx; **no** reintentar 4xx.
   - Orquestación: exportar/importar workflows n8n (JSON) como artefactos de CI.

---

7) 🧱 Modelo de Datos



8) Casos de uso

✨ CU-1 Registrar versión

- Endpoint: POST /api/versiones
- Pre: (cliente,nombre,build) no existe
- Post: Version en `Draft` ; auditoría creada
- Errores: 409 duplicado
- Logs: correlationId

✨ CU-2 Adjuntar artefactos

- Endpoint: POST /api/versiones/{id}/artefactos
- Pre: versión `Draft|Ready`
- Post: Artefacto vinculado con md5; auditoría
- Errores: 404 versión, 422 tipo/rama inválidos

✨ CU-3 Validar versión

- Endpoint: POST /api/versiones/{id}/validar
- Pre: `Draft` con artefactos/MD5
- Reglas: naming, mínimo por rama, consistencia `buildYYYYMMDD`

- Post: estado → `Ready` ; auditoría
- Errores: 422 reglas

✨ CU-4 Publicar versión (con Worker)

- Endpoint: POST /api/versiones/{id}/publicar
- Pre: `Ready`
- Acción: encolar jobs (`COPY_ARTIFACTS` , `COMPUTE_MD5` , `GEN_OUTBOX`)
- Post: verificación outputs; estado → `Published` ; auditoría
- Errores: 404/409/422; 502/503 (externo)

✨ CU-5 Notificar interesados

- Disparo: al pasar a `Published`
- Local (default): generar `.eml` + `release-notes.md` en `/data/outbox`
- Graph (opc.): crear/actualizar borrador en Outlook (mismo hilo)
- Post: borrador referenciado en `Borrador`

✨ CU-6 Integración OneDrive (opc.)

- Acción: subir por chunks y leer MD5 remoto (adapter Graph)
- Post: URL/ruta en artefacto; auditoría
- Errores: 5xx externos + reintentos

✨ CU-7 Integración con repos (opc.)

- Acción: asociar hash/URL de artefacto/repositorio
- Post: trazabilidad en `Version