Plan Operativo LOD1 – v1.1

Clasificación Asistida de Patologías con SAM-3

0.1 Objetivo de la Fase 0 (LOD 1)

1. Tener un repositorio listo para desarrollo (estructura, entornos, dependencias, scripts de arranque).

2. Poder arrancar el backend y comprobar de forma fiable:

- que **SAM-3 carga**,
- que la **GPU/VRAM** es usable (o caer a CPU de forma controlada),
- que el sistema puede ejecutar **una inferencia mínima** (warm-up) sin errores.

3. Ejecutar un **preanálisis rápido del dataset** (conteo + consistencia de atributos críticos).

4. Disponer de una **pantalla única** (“Estado del sistema”) que muestre salud y avisos.

0.2 Estructura propuesta del repositorio (monorepo modular)

Recomendación: **monorepo** con backend + frontend + herramientas, y con dependencias ML aisladas. Ejemplo:

```
Proyecto_IA/
  README.md
  .gitignore
  .gitattributes

  apps/
    backend/
      src/
        api/
        core/
        inference/
        jobs/
        datasets/
        stats/
        storage/
      tests/
      pyproject.toml
      .env.example

    frontend/
      src/
      public/
      package.json
      vite.config.ts (o next.config.js)
      .env.example

  scripts/
    dev/
      run_backend.ps1
      run_backend.sh
      run_frontend.ps1
      run_frontend.sh
    setup/
      check_env.py
      download_models.py
      dataset_scan.py

  data/
    datasets/.gitkeep
    cache/.gitkeep
```

```
checkpoints/
  sam3/.gitkeep

vendor/
  sam3/            (submódulo o clon del repo oficial)
  sam3_extras/     (si aparece tooling auxiliar oficial)

docs/
  lod0/
  lod1/
  adr/             (Architecture Decision Records)
  api/             (OpenAPI)

logs/.gitkeep
```

Por qué así

- `apps/backend` y `apps/frontend` quedan independientes, pero versionados juntos.
- `vendor/` sirve para clonar repos externos **sin ensuciar** la base del proyecto.
- `checkpoints/` y `data/` están **fuera de control de versiones** (se ignoran), pero con `.gitkeep` para que exista el árbol.
- `scripts/setup` concentra lo repetible: checks, descargas, escaneo dataset.

0.3 Repositorios a clonar y ubicación (vendor/)

En LOD 1 Fase 0, el único clon crítico es el del **repo oficial de SAM-3** (y solo si no lo traemos como dependencia pip).

Opciones recomendadas (elige una y la estandarizamos):

Submódulo git (recomendado en repos privados)

- Ventajas: versiones fijadas, reproducible, limpio.
- Ruta: `vendor/sam3`

usar **submódulo** para fijar commit, y exponer un wrapper en `apps/backend/src/inference/`.

0.4 Pesos / checkpoints necesarios (checkpoints/sam3)

En Fase 0 no necesitamos “todos los pesos”, solo **uno** que sea el baseline del PoC.

Estrategia LOD 1:

- Definir en configuración un `SAM3_MODEL_ID` (por ejemplo “small/large”), y una ruta cache local:
    - `checkpoints/sam3/<model_id>/...`
- Implementar un script `scripts/setup/download_models.py` que:
    - Verifique si existe el checkpoint.
    - Si no existe, lo descargue (requiere credenciales si aplica).
    - Registre en log: `model_id`, `fecha`, `hash`, `tamaño`.

Decisión importante (ADR): en LOD 1 fijamos *una* variante (p.ej. “base/large”) y no abrimos compatibilidad múltiple hasta tener el pipeline estable.

0.5 Configuración: unificada, explícita y auditable

Propuesta: configuración por entorno en backend:

- `apps/backend/.env` (no versionado; se versiona `.env.example`)
- `apps/backend/src/core/config.py` (lee env y valida)

Variables mínimas Fase 0:

- `APP_ENV=dev|prod`
- `DATASETS_ROOT=/ruta/por/defecto` (opcional)
- `CHECKPOINTS_DIR=../../checkpoints/sam3`
- `SAM3_MODEL_ID=...`
- `DEVICE_PREFERENCE=auto|cuda|cpu`
- `TARGET_LONG_SIDE=1024`
- `WARMUP_IMAGE=/ruta/imagen/test` (o usar una imagen “fixture”)
- `LOG_LEVEL=INFO`

0.6 Dataset: análisis y preprocesamiento sencillo (Fase 0)

Objetivo

Antes de inferir, necesitamos saber:

- cuántas imágenes hay,
- formatos,
- dimensiones,
- orientación EXIF,
- si hay outliers (muy pequeñas, muy grandes, rotadas, corruptas),
- si la distribución sugiere que el pipeline debe normalizar más agresivamente.

Script propuesto: `scripts/setup/dataset_scan.py`

Salida sugerida (JSON + print amigable):

- `total_images`
- `extensions_count` (jpg/png/…)
- `corrupt_images_count` + listado (primeros N)
- `dimensions_histogram` (top 10 resoluciones)
- `aspect_ratio_stats` (min/max/percentiles)
- `exif_orientation_count` (cuántas con rotación)
- `color_mode_count` (RGB/L/CMYK si aparece)
- muestreo aleatorio de N imágenes para métricas rápidas

Importante: esto no “preprocesa” aún guardando nuevas imágenes (eso se decide en Fase 1). Aquí solo **audita** y recomienda.

Reglas de consistencia “críticas” (para tu dominio)

- EXIF Orientation: si hay muchas rotadas, hay que normalizar antes de inferencia.
- Dimensiones muy dispares: confirmar si reescalado a `TARGET_LONG_SIDE` es suficiente.
- Formatos raros/CMYK: convertir en carga.

BACKEND — Módulo de Inferencia SAM-3 (Fase 0)

0.7 API mínima de Fase 0

Endpoints mínimos:

- `GET /api/v1/health`
- (Opcional) `POST /api/v1/system/warmup` si quieres dispararlo manualmente desde UI

El `GET /health` debe devolver:

- `sam3_model_loaded: bool`
- `sam3_model_version / model_id`
- `device_selected: cuda|cpu`
- `gpu: available, name, total_vram, free_vram`
- `target_long_side`
- `warmup: status, last_run, latency_ms`
- `errors: []` (si aplica)

0.8 Pasos del módulo de inferencia (LOD 1 propositivo)

Paso 1 — Selección de dispositivo (auto/cuda/cpu)

Orden recomendado:

1. Si `DEVICE_PREFERENCE=cpu` → CPU.
2. Si `cuda` y hay GPU utilizable → CUDA.
3. Si `auto`:
    - si CUDA disponible y VRAM suficiente → CUDA,
    - si no → CPU con warning.

Warning no debe bloquear (salvo que el usuario haya forzado `cuda`).

Paso 2 — Carga de checkpoints (lazy + cache)

- Verifica si el checkpoint está en `CHECKPOINTS_DIR`.
- Si no está:
    - devuelve error controlado con instrucción clara (“descargar pesos”).
- Carga el modelo en memoria y marca `sam3_model_loaded=true`.

Paso 3 — Config de resolución objetivo

- Política: **“long side = TARGET_LONG_SIDE”**, manteniendo ratio.
- Se aplica en un preprocesador único (reutilizable en Fase 1/2/3).
- Se registra en `health`.

Paso 4 — Warm-up (inferencias de prueba)

Es una **prueba controlada** antes de lanzar procesos largos.

Diseño recomendado

- Input: una imagen pequeña de test (o una del dataset si el usuario ya lo registró).
- Output: no importa el resultado visual; importa:
    - éxito/fracaso,
    - latencia,
    - consumo aproximado VRAM (si lo medimos).

El resultado del warm-up se muestra en la pantalla “Estado del sistema”.

FRONTEND — Pantalla “Estado del sistema” (Fase 0)

0.9 UX: flujo lineal (sin pestañas)

Dado que tu proceso es lineal, propongo una UI tipo **wizard** con rutas, pero con navegación restringida:

- `/system/status` (Fase 0)
- `/datasets` (Fase 0/1)
- (las demás fases se desbloquean después)

En Fase 0 solo implementamos `/system/status`.

0.10 Estructura de la pantalla

Componentes mínimos:

A) Card “Modelo SAM-3”

- Estado: `Cargado / Error / No inicializado`
- Modelo: `SAM3_MODEL_ID`
- Versión/commit (si está disponible)
- Resolución objetivo: `TARGET_LONG_SIDE`

Acción:

- Botón “Ejecutar warm-up” (si lo haces manual)

B) Card “Hardware”

- Dispositivo seleccionado: `cuda/cpu`
- GPU:
    - nombre,
    - VRAM total/libre
- CPU: cores (opcional), RAM (opcional)
- Mensaje de degradación si está en CPU

C) Panel de alertas (muy visible)

- “GPU no detectada” → warning
- “VRAM inferior a la recomendada” → warning (con recomendación concreta)
- “Modelo no descargado / no encontrado” → error con CTA (“Descargar pesos”)

D) Sección “Diagnóstico rápido”

- Warm-up:
  - último estado
  - latencia
  - timestamp

- Logs recientes (últimas 20 líneas del backend, opcional pero muy útil en PoC)

0.11 Contrato FE–BE (Fase 0)

El frontend obtiene el estado del sistema exclusivamente a través del backend mediante el endpoint:

`GET /api/v1/health`

Este endpoint actúa como **fuente única de verdad** sobre:

- estado de carga del modelo SAM-3,
- dispositivo efectivo (CPU/GPU),
- disponibilidad y estado de la GPU (VRAM),
- configuración activa (modelo y resolución),
- resultado del warm-up (si se ha ejecutado).

Uso desde el frontend

- Se llama **al cargar la pantalla “Estado del sistema”**.
- Se vuelve a llamar **tras acciones que cambian el estado** (p. ej. ejecución de warm-up).
- Se ofrece **refresco manual** por parte del usuario.
- **No se implementa polling periódico** en estado estable.
- De forma opcional, puede habilitarse un **refresco temporal** durante estados transitorios (carga del modelo o warm-up), que se desactiva automáticamente al finalizar.

Principios

- El frontend **no infiere estado** ni mantiene lógica duplicada.
- La UI **renderiza únicamente los datos recibidos**.
- El contrato se mantiene **mínimo, estable y orientado a diagnóstico**.

0.12 Entregables concretos de la Fase 0 (para controlar el alcance)

Entregables repositorio

1. Estructura de carpetas implementada.
2. `.gitignore` correcto (data/checkpoints/logs).
3. `docs/adr/0001-repo-structure.md` (decisiones clave).

Entregables backend

4. `GET /api/v1/health` funcional.
5. Módulo `inference/sam3_engine.py` con:
    - selección de device,
    - carga modelo,
    - preprocesado,
    - warm-up.

Entregables dataset tools

6. `dataset_scan.py` que genere `dataset_report.json` y resumen en consola.

Entregables frontend

7. `/system/status` mostrando health real + alertas + botón warm-up (si aplica).

LOD 1 — Fase 1

Clasificación Jerárquica I (Nivel 1 · Detección Masiva)

Objetivo

Realizar una **preclasificación masiva y genérica del dataset**, etiquetando imágenes completas con **conceptos amplios** (p. ej. *Fachada*, *Cubierta*), que sirva como **filtro estructural** para las fases posteriores.

---

1. Conector con Fase 0 (condiciones de entrada)

La Fase 1 **solo puede ejecutarse** si se cumplen las siguientes condiciones, verificadas desde el backend:

- El modelo **SAM-3 está cargado** (`model_state = loaded`).
- El dispositivo efectivo (GPU o CPU) está definido.
- El dataset seleccionado:
    - ha sido registrado,
    - ha pasado el escaneo básico (conteo, imágenes accesibles),
    - no presenta errores críticos.

Si alguna condición no se cumple:

- el backend rechaza el lanzamiento del job,
- el frontend muestra un mensaje claro y accionable.

Esto evita lanzar procesos largos en estados inválidos.

---

2. Naturaleza del proceso (expectativa clara para el usuario)

- La clasificación de Nivel 1 es un **proceso largo y secuencial**.
- Puede durar **minutos u horas**, dependiendo de:
    - tamaño del dataset,
    - dispositivo (GPU/CPU),
    - batch size,
    - número de conceptos.

- Está diseñado para:
  - **no bloquear la UI**,
  - **ser cancelable**,
  - **poder reanudarse**.

Desde el punto de vista UX, el usuario debe entender que:

> “Este es un proceso de fondo, puedes dejarlo correr y volver a consultar su estado.”

---

3. Backend — Inferencia Nivel 1 (LOD 1)

Flujo de inferencia por imagen

Para cada imagen del dataset:

1. **Carga y preprocesado**
   - Corrección de orientación (EXIF).
   - Reescalado a `TARGET_LONG_SIDE`.
   - Normalización.

2. **Ejecución SAM-3 (PCS)**
   - Aplicación de la lista de prompts de Nivel 1.
   - Obtención de:
     - máscaras,
     - bounding boxes,
     - score de confianza.

3. **Persistencia**
   - Inserción de resultados en `regions` con `level = 1`.
   - Asociación con `concept_id`, `image_id` y `job_id`.

---

4. Backend — Sistema de Jobs (Nivel 1)

Definición del job

- `job_type = level1_inference`

Parámetros

- `dataset_id`
- `concept_ids` (Nivel 1)
- `user_confidence_threshold`
- `batch_size`
- (opcional) `max_images` para pruebas

Estados del job

- `pending`
- `running`
- `completed`
- `failed`
- `cancelled`

Reanudación

- El job mantiene un **cursor por `image_id`**.
- Si el proceso se interrumpe:
    - puede reanudarse desde el último punto consistente,
    - sin repetir imágenes ya procesadas.

---

5. Progreso, logs y feedback (clave en Fase 1)

Progreso cuantitativo (obligatorio)

El backend expone en todo momento:

- `total_images`
- `processed_images`
- `progress_percent`
- `estimated_remaining_time` (estimación simple, no crítica)

Esto permite al frontend mostrar:

- barra de progreso,
- porcentaje,
- ETA aproximado.

Log operativo resumido (no técnico)

Además del progreso numérico, el backend mantiene un **log de job de alto nivel**, por ejemplo:

- “Cargando batch 120–128”
- “Procesadas 10.000 / 120.000 imágenes”
- “GPU VRAM cercana al límite, reduciendo batch”
- “Job cancelado por el usuario”

El frontend:

- muestra las **últimas N líneas** (no un log técnico completo),
- las usa como **feedback psicológico** de avance.

Objetivo: evitar la sensación de “esto está colgado”.

---

6. Backend — Estadísticas Nivel 1

Al finalizar (o incrementalmente), se calculan:

- Nº de imágenes con **al menos una región** por concepto.
- Distribución de scores por **buckets de confianza**, definidos a partir de:
    - `User_Confidence` (mínimo aceptable),
    - límite superior fijo (0.9).

Estas estadísticas:

- no bloquean el job,
- se recalculan si es necesario,
- alimentan directamente la UI.

---

7. Frontend — Pantalla “Clasificación Nivel 1”

Entrada

- Selector de dataset.
- Lista editable de conceptos Nivel 1 (prompts).
- Configuración básica, con umbral de confianza y batch size (si se expone).

Ejecución

- Botón **“Lanzar clasificación Nivel 1”**.
- El usuario pasa a una vista de **job activo**.

Durante el proceso

- Barra de progreso (% + ETA).
- Estado textual del job.
- Panel de **log resumido** (stream o refresco manual).
- Controles: Cancelar job y reanudar (si aplica).

Resultados (cuando hay datos)

- Tabla por concepto:
    - Nº de imágenes afectadas.

- Gráfico de barras por buckets de confianza.
- Galería de ejemplos (thumbnails) por bucket.

LOD 1 — Fase 2

Clasificación Jerárquica II (Nivel 2 · Subcomponentes)

Objetivo

Refinar las detecciones de Nivel 1 identificando **subcomponentes arquitectónicos** (balcones, aleros, cornisas, etc.) **dentro de una clase padre concreta**, reduciendo ruido y aumentando precisión contextual.

---

1. Conector con Fase 1 (condiciones de entrada)

La Fase 2 **depende explícitamente** de la Fase 1 y solo puede ejecutarse si:

- Existe al menos un **concepto de Nivel 1** con regiones detectadas (p. ej. *Fachada*).
- El usuario selecciona:
  - un `parent_concept_id` (Nivel 1),
  - un umbral mínimo de confianza heredado (`confidence_level1 ≥ threshold`).
- El backend confirma que existen regiones válidas de Nivel 1 que cumplen el criterio.

Si no hay regiones elegibles:

- el job no se lanza,
- el frontend informa claramente (“No hay fachadas con confianza suficiente”).

---

2. Naturaleza del proceso (expectativa de uso)

- La Fase 2 es **selectiva**, no masiva:
  - se ejecuta solo sobre un subconjunto del dataset,
  - delimitado por detecciones previas.
- Es más rápida que la Fase 1, pero **sigue siendo asíncrona**.
- Está pensada para:
  - exploración iterativa,
  - ajuste de prompts,
  - validación visual frecuente.

Desde UX:

> “Estamos refinando dentro de lo ya detectado, no empezando de cero.”

---

3. Backend — Inferencia Nivel 2 (LOD 1)

Scope de procesamiento

Para un job de Nivel 2:

- Se seleccionan **regiones de Nivel 1** que:
  - pertenecen a `parent_concept_id`,
  - cumplen `confidence_level1 ≥ min_confidence_level1`.

Cada región de Nivel 1 actúa como **contexto espacial**.

Estrategia de inferencia (decisión LOD 1)

Estrategia base (recomendada para PoC):

- Generar **crops** a partir de las máscaras/bboxes de Nivel 1.
- Ejecutar PCS de SAM-3 sobre cada crop con prompts de Nivel 2.

Ventajas:

- Menor coste computacional.
- Menos ruido semántico.
- Mejor alineación “subcomponente dentro de componente”.

Nota LOD 1:

- El modelo de datos conserva **máscaras y bboxes completas**, de modo que en el futuro puede probarse inferencia sobre imagen completa sin romper compatibilidad.

Persistencia de resultados

- Cada detección se almacena en `regions` con:
  - `level = 2`,
  - referencia al `image_id`,
  - referencia indirecta al `parent_concept_id`,
  - máscara, bbox y score.

- Se mantiene trazabilidad completa con el `job_id`.

---

4. Backend — Sistema de Jobs (Nivel 2)

Definición del job

- `job_type = level2_inference`

Parámetros

- `dataset_id`
- `parent_concept_id` (Nivel 1)
- `concept_ids` (Nivel 2)
- `min_confidence_level1`
- `batch_size` (opcional)

Estados

- `pending`
- `running`
- `completed`
- `failed`
- `cancelled`

Progreso

El progreso se expresa en términos de:

- Nº de **regiones de Nivel 1 procesadas**,
- no de imágenes totales del dataset.

Esto es clave para que el feedback tenga sentido para el usuario.

---

5. Backend — Estadísticas Nivel 2

Las estadísticas se calculan **en relación al contexto**, no en absoluto:

- Nº de regiones de Nivel 1 procesadas.
- Nº de regiones de Nivel 2 detectadas por concepto.
- Distribución de scores por buckets de confianza.

Esto permite responder preguntas como:

- "¿Cuántos balcones aparecen dentro de fachadas detectadas?"
- "¿En qué rango de confianza se concentran?"

---

6. Frontend — Pantalla "Clasificación Nivel 2"

Entrada

Formulario claro y acotado:

- Selector de dataset.
- Selector de concepto de Nivel 1 (p. ej. *Fachada*).
- Lista editable de conceptos de Nivel 2 (prompts).
- Umbral mínimo de confianza heredado de Nivel 1

Ejecución

- Botón **"Lanzar clasificación Nivel 2"**.
- Vista de job activo con progreso (regiones procesadas) y su estado textual.

Resultados

Panel estructurado en tres niveles:

1. **Tabla por concepto Nivel 2**

- Nº de imágenes afectadas.
- Nº total de regiones detectadas.
- Distribución de confianza.

2. **Muestras visuales**

- Imagen base.
- Máscara de Nivel 1 (contexto).
- Regiones de Nivel 2 superpuestas.

3. **Validación rápida**

- Permite detectar falsos positivos, prompts mal definidos, conceptos redundantes.

LOD 1 — Fase 3

Clasificación Jerárquica III (Nivel 3 · Patologías por Ejemplar)

Objetivo

Detectar **patologías constructivas específicas** (grietas, humedades, desprendimientos, corrosión, etc.) mediante **búsqueda por similitud visual**, a partir de **ejemplos marcados por el usuario**, apoyándose en la estructura jerárquica creada en los Niveles 1 y 2.

Esta fase introduce el **conocimiento experto humano** de forma explícita en el sistema.

---

1. Conector con Fases 1 y 2 (condiciones de entrada)

La Fase 3 **no es independiente** y se apoya en la jerarquía previa:

- Deben existir:
    - detecciones válidas de **Nivel 1** (p. ej. *Fachada*),
    - opcionalmente de **Nivel 2** (p. ej. *Balcones*, *Aleros*).

- El usuario define explícitamente el **scope de búsqueda**, que puede ser:
    - todo el dataset,
    - o un subconjunto filtrado por Nivel 1 y/o Nivel 2.

Este filtro previo es **clave** para:

- reducir ruido,
- acelerar inferencia,
- aumentar la precisión semántica.

---

2. Naturaleza del proceso

Detección por similitud guiada por ejemplos usando PCS de SAM-3.

Desde UX, el mensaje implícito es:

> “Enséñame qué es esta patología y la buscaré donde tenga sentido.”

---

3. Backend — Conceptos de Nivel 3 (LOD 1)

Definición del concepto

Un concepto de Nivel 3 representa **una patología concreta** y se define como:

- `level = 3`
- `type = exemplar`
- Nombre descriptivo (p. ej. *Grieta vertical en fachada*)
- Asociado a uno o varios **ejemplos visuales**

Cada concepto:

- es **independiente**,
- puede evolucionar añadiendo o quitando ejemplos,
- no requiere reentrenar modelos.

---

4. Backend — Ejemplos (Examples)

Cada ejemplo consiste en:

- una imagen del dataset,
- un **bounding box** marcado por el usuario que delimita la patología,
- un comentario opcional (contexto técnico).

Los ejemplos:

- son explícitos y auditables,
- representan conocimiento experto,
- no se mezclan entre conceptos.

---

5. Backend — Inferencia Nivel 3 (Exemplar PCS)

Scope de inferencia

Para un job de Nivel 3:

- Se define un **conjunto objetivo de imágenes/regiones**, filtrado por:
    - Nivel 1 (obligatorio en práctica),
    - Nivel 2 (altamente recomendado).
- Esto convierte la búsqueda en **focalizada**, no global.

Flujo de inferencia

Para cada ejemplo activo del concepto:

1. Se ejecuta PCS contra el subconjunto definido.
2. Para cada imagen/región candidata:
    - se generan **regiones detectadas**,
    - se asigna un **score por ejemplo**.
3. Se almacenan:
    - la región (level = 3),
    - la relación *(region, example)* en `evidence`.

El resultado no es una única puntuación, sino **múltiples evidencias parciales**.

---

6. Backend — Agregación de Evidencia (decisión clave LOD 1)

Las evidencias se combinan para obtener un **score agregado por región**, que representa la probabilidad de que esa región sea realmente la patología buscada.

- Se utiliza una agregación probabilística (no un simple promedio).
- El score agregado:
    - se recalcula cuando cambian los ejemplos,
    - se guarda asociado a la región.

Este score es la **base objetiva** para la Fase 4 (automatización).

---

7. Backend — Sistema de Jobs (Nivel 3)

Definición del job

- `job_type = level3_inference`

Parámetros

- `dataset_id`
- `concept_id` (Nivel 3)
- `example_ids` activos
- `scope` (filtros por Nivel 1 / Nivel 2)

Estados

- `pending`
- `running`
- `completed`
- `failed`
- `cancelled`

Progreso

El progreso se expresa como:

- Nº de imágenes/regiones evaluadas dentro del scope,
- no como porcentaje del dataset total.

Esto evita métricas engañosas.

---

8. Backend — Estadísticas Nivel 3

Para cada patología se exponen:

- Histograma de `aggregated_score`.
- Nº de regiones por rangos de confianza:
    - alta,
    - media,
    - baja.

Estas estadísticas permiten:

- evaluar la calidad del concepto,
- decidir si hacen falta más ejemplos,
- anticipar el nivel de automatización posible.

---

9. Frontend — Pantalla “Búsqueda por Ejemplo (Nivel 3)”

Flujo guiado (lineal)

Paso 1 — Definir patología

- Nombre descriptivo y claro.

Paso 2 — Añadir ejemplos

- Selector de imagen del dataset.
- Herramienta visual de marcado (bounding box).
- Lista de ejemplos actuales (miniaturas).

Paso 3 — Definir scope y lanzar búsqueda

- Todo el dataset o filtrado por Nivel 1 / Nivel 2.
- Botón “Lanzar job Nivel 3”.

Paso 4 — Resultados iniciales

- Galería de regiones detectadas con máscaras y scores.

- Posibilidad de marcar: “claramente correcto” o “falso positivo”.

Este feedback no reentrena el modelo, pero **informa la decisión humana**.

LOD 1 — Fase 4

Automatización, Evidencia y Validación Masiva

Objetivo

Transformar las detecciones de Nivel 3 (patologías por ejemplar) en **etiquetado masivo controlado**, aplicando **reglas automáticas basadas en evidencia** y ofreciendo al usuario mecanismos eficientes de **revisión y validación** allí donde la confianza no es suficiente.

Esta fase convierte el sistema en una **herramienta productiva**, no solo exploratoria.

---

1. Conector con la Fase 3 (condiciones de entrada)

La Fase 4 se apoya exclusivamente en los resultados consolidados del Nivel 3:

- Existen uno o varios **conceptos de Nivel 3** con:
  - regiones detectadas,
  - `aggregated_score` calculado por región.

- El usuario selecciona explícitamente:
  - una patología concreta,
  - los criterios de decisión (umbrales).

Sin evidencia agregada válida, la Fase 4 **no puede ejecutarse**.

---

2. Naturaleza del proceso (qué hace el sistema)

Se aplican **reglas deterministas** sobre resultados ya calculados.

---

3. Backend — Reglas de Decisión (LOD 1)

Parámetros por patología (concepto Nivel 3)

Para cada concepto, el usuario define:

- `threshold_accept`
  - score mínimo para **aceptación automática**.
- `threshold_reject`
  - score máximo para **rechazo automático**.
- `sample_size_pending`
  - tamaño de muestra de la **zona gris** para revisión manual.

Entre ambos umbrales se define una **zona intermedia** (*pending_review*).

Aplicación de reglas

Para cada región del concepto:

- Si `aggregated_score ≥ threshold_accept`
  → `status = accepted_auto`
- Si `aggregated_score ≤ threshold_reject`
  → `status = rejected_auto`
- En caso contrario
  → `status = pending_review`

Estos estados se almacenan y son **reproducibles y auditables**.

---

4. Backend — Muestreo Estratificado

Sobre las regiones en `pending_review`:

- Se genera un **muestreo representativo**, no aleatorio puro.
- El muestreo se distribuye por rangos de score para evitar sesgos.
- El objetivo es:
    - validar rápidamente la frontera de decisión,
    - detectar falsos positivos sistemáticos.

Este muestreo es el principal punto de interacción humana en esta fase.

---

5. Backend — Informes y Cobertura

El backend calcula y expone:

- Nº total de regiones:
    - aceptadas automáticamente,
    - rechazadas automáticamente,
    - pendientes de revisión.

- Cobertura efectiva:
    - porcentaje del dataset ya clasificado,
    - volumen restante sin decisión final.

Estos datos permiten al usuario decidir:

- si los umbrales son adecuados,
- si necesita más ejemplos (volver a Fase 3),
- o si el resultado es suficientemente fiable.

---

6. Backend — Exportación de Resultados

Una vez aplicadas las reglas, el sistema permite exportar:

- Listados estructurados (CSV / JSON) con:
    - path de imagen,
    - bounding box,
    - score agregado,
    - estado final.

- Opcionalmente:
    - un dataset limpio para futuros entrenamientos,
    - o para integración con otros flujos (informes, BIM, etc.).

---

7. Frontend — Pantalla “Validación Masiva y Reglas”

Entrada

- Selector de patología (Nivel 3).
- Controles claros para:
    - `threshold_accept`,
    - `threshold_reject`.

Visualización de impacto

Al modificar los umbrales, la UI muestra inmediatamente:

- Nº de regiones en cada estado:
    - aceptadas,
    - rechazadas,
    - pendientes.
- Histograma de `aggregated_score` con marcas visuales de los umbrales.

Esto convierte la decisión en **visual y comprensible**, no abstracta.

Revisión de la zona gris

- Galería de muestras seleccionadas.
- Acciones rápidas:
    - aceptar en bloque,
    - marcar falsos positivos,
    - descartar.

La revisión está pensada para ser **rápida y representativa**, no exhaustiva.

---

8. Acciones globales

La pantalla ofrece acciones explícitas y controladas:

- “Aplicar reglas a todo el dataset”
- “Recalcular estados tras añadir nuevos ejemplos”
    - (vuelve a Fase 3 y reevalúa Fase 4)