E4.3 Informe de resultados del caso de uso simulado en un escenario controlado

GOBIERNO DE ESPAÑA
VICEPRESIDENCIA TERCERA DEL GOBIERNO
MINISTERIO PARA LA TRANSICIÓN ECOLÓGICA Y EL RETO DEMOGRÁFICO
Fundación Biodiversidad
Plan de Recuperación, Transformación y Resiliencia
Financiado por la Unión Europea
NextGenerationEU

Contenido

1. Introducción

1.1 Objetivo

Este documento corresponde al entregable:

E4.3 - Informe de resultados del caso de uso simulado en un escenario controlado

Refleja los trabajos realizados y los resultados alcanzados durante la ejecución de la actividad:

A4.1 Módulo de gestión de información, elaboración de directrices y criterios de planificación ambiental, A4.2 Tecnologías de análisis de visualización de datos maisvos para la predicción y protección de aves, A4.3 Módulo de gestión de información, elaboración de directrices y criterios de planificación ambiental.

Esta tarea se encuadra dentro de LA ACTIVIDAD A4 y la línea de investigación:

A4. Sistema de gestión de la información de aves y tecnologías de visualización, cuyo objetivo se centra en investigar y diseñar tecnologías de análisis y visualización de datos masivos para la predicción y protección de aves que permita optimizar la gestión de la información y elaborar directrices y criterios de planificación ambiental.

1.2 Alcance

Este documento se encuentra en la versión 1.0, y aunque está sujeta a nuevas evoluciones si en las siguientes fases del proyecto se hacen necesarios ajustes, la investigación aquí plasmada trata de presentar el trabajo realizado para la consecución de los objetivos de la tarea de diseñar una plataforma de IA explicable para optimizar la evaluación de espacios destinados a ser parques eólicos.

2 Diseño de conectores para interoperabilidad de componentes

El objetivo de esta actividad es estudiar es diseñar e implementar los diferentes conectores para la interoperabilidad entre los componentes que conforman la plataforma de forma armonizada.

2.1 Visión General y Arquitectura de la Plataforma

La ***A4. Sistema de gestión de la información de aves y tecnologías de visualización***, implementa una plataforma full stack que consume los resultados del ***A3. Machine learnig para la detección y monitorización de aves*** y los expone de forma controlada a los perfiles que toman decisiones. La plataforma integra tres fuentes: i) monitorización con sensores IoT (cámaras y micrófonos), ii) ciencia ciudadana (Xeno canto, eBird) y iii) contexto territorial (Mapa Eólico Ibérico y zonas de exclusión de IDECYL).

La arquitectura elegida es híbrida Edge + Cloud. En el borde (Jetson u otro dispositivo acelerado) se ejecuta la detección en tiempo real; en la nube residen ingesta, almacenamiento, agregación y visualización. El reparto resuelve restricciones de ancho de banda y latencia: los streams no se trasladan a la nube; lo que viaja son mensajes ligeros con resultados (detecciones, conteos, heatmaps) sobre MQTT. Este canal asíncrono tolera cortes de red y evita pérdidas.

El flujo de datos inicia en el Edge: el vídeo RTSP de la cámara se procesa con los modelos de detección. El dispositivo publica resultados estructurados en Mosquitto. En la nube, el servicio

de ingesta, suscrito a los tópicos, valida y persiste en PostgreSQL. El backend expone estos datos por API y el frontend los representa en mapa, series y paneles.

El vídeo en vivo se captura en RTSP desde la cámara y se entrega al servicio MediaMTX, que lo redistribuye como HLS. El uso de HLS garantiza compatibilidad con navegadores sin extensiones.

2.2 Arquitectura en la nube y microservicios

La plataforma se despliega como microservicios contenerizados coordinados por redes internas. Nginx actúa como puerta de entrada, termina TLS y enruta al frontend (React, construida con Vite) y al backend (Node.js). El backend autentica, autoriza por rol y compone respuestas agregando datos de servicios internos.

El ai4birds coordinate service (Flask) aplica control de acceso: valida JWT, registra acceso y reenvía llamadas autorizadas al ai4birds ingest service. El ingest service escucha MQTT para recepcionar detecciones y heatmaps, persiste en la base operativa y ejecuta tareas periódicas con Celery para consultar Xeno canto y eBird, obtener el Mapa Eólico Ibérico para coordenadas y cargar capas de IDECYL. La API está documentada con Flask RestX/Swagger.

El almacenamiento se estructura en tres instancias de PostgreSQL: configuración (usuarios, cámaras, roles), ingesta operativa (segmentos, estadísticas, observaciones, referencias de heatmap) y análisis (vistas materializadas y agregados temporoespaciales). Redis hace de broker de Celery y de caché de resultados. MediaMTX recibe RTSP y publica HLS; su API permite dar de alta orígenes sin tocar componentes adyacentes.

2.3 APIs y flujos de acceso a los datos

La API de coordinación es la puerta segura de uso por la interfaz. El frontend adjunta el JWT en cada llamada. El coordinador verifica token y permisos y, si procede, reenvía al servicio de ingesta. Esta consulta la base operativa y devuelve JSON. Con este patrón, al abrir el detalle de una cámara, la plataforma recupera detecciones del intervalo, estadísticas por especie y el descriptor del heatmap más reciente; el frontend los presenta en sus vistas.

La API de ingesta ofrece los recursos de explotación: segmentos filtrables por cámara, ventana y confianza; estadísticas agregadas por especie y franja horaria; descriptores de heatmap con image_url y bounds; resultados combinados de Xeno canto y eBird; y datos de viento/exclusión para contexto. La cadena de responsabilidades queda separada: Edge detecta y publica; Ingesta valida, persiste y sirve; Coordinación asegura; BFF compone; Frontend presenta.

2.4 Seguridad, cumplimiento y operatividad

Todas las comunicaciones externas viajan bajo TLS. La autenticación usa JWT y la autorización se establece por rol y, cuando aplica, por ámbito. Se aplica minimización de datos y ocultación espacial en perfiles sin permiso para coordenadas precisas. La resiliencia se mantiene con rate limiting, circuit breakers para proveedores lentos y colas (MQTT/Celery) que amortiguan picos y cortes. La observabilidad se apoya en registros estructurados, métricas de rendimiento y paneles de salud.

3 Implantar y testear la plataforma en un caso de uso controlado

3.1 Despliegue y validación en escenario controlado

El entorno de referencia usa Docker Compose: Nginx, frontend React, backend Node.js, coordinación e ingesta (Flask), MediaMTX, Mosquitto, Redis y tres PostgreSQL.

Las redes internas aíslan el tráfico y solo el gateway publica 80/443. Los datos y configuraciones críticas se alojan en volúmenes persistentes con gestión automática de certificados TLS. Sobre este entorno se ejecutan pruebas unitarias en Python/JS, integración con Postman y publicaciones MQTT simuladas, además de recorridos E2E desde login hasta la visualización de heatmaps.

3.2 Pruebas funcionales para la depuración

Se realizaron pruebas funcionales, unitarias y de integración en los módulos que nuclean la lógica central de la plataforma, con un foco principal en los servicios de Python/Flask (API de Ingesta y Coordinación) y en el flujo de ingesta de datos a través de MQTT.

3.2.1 Pruebas de API (Unitarias y de Integración HTTP)

Para validar los endpoints REST de los servicios ai4birds-ingest-service y ai4birds-coordinate-service, se utilizó la librería requests-mock.

- Simulación de Peticiones: requests-mock permite interceptar las llamadas HTTP salientes, simulando las respuestas de los endpoints.
- Validación de Endpoints: Se crearon pruebas para cada endpoint (ej. /ebird, /xenocanto, /windmap, /device-status, etc.).

Escenarios de Prueba

- Prueba de Éxito: Se verifica que el endpoint responde con el código de estado esperado (HTTP 200 OK).
- Prueba de Errores: Se comprueba que el endpoint maneja correctamente los errores, devolviendo códigos 404 (No Encontrado) o 500 (Error Interno) cuando es apropiado.
- Prueba de Parámetros: Se utiliza la función @pytest.mark.parametrize para enviar diferentes cuerpos de solicitud (JSON) a endpoints que lo requieren (como /windmap o /device-status) y validar su procesamiento.
- Prueba de Autenticación: Se valida que el header Authorization: Bearer <TOKEN> se construye y envía correctamente en las peticiones que lo requieren.

```python
import pytest
import requests_mock
import requests
from config_test import BASE_URL, TOKEN, EXPECTED_STATUS_200_500, EXPECTED_STATUS_200_500_404

@pytest.fixture
def mock_requests():
    with requests_mock.Mocker() as mock:
        yield mock

def get_headers():
    headers = {"Content-Type": "application/json"}
    if TOKEN:
        headers["Authorization"] = f"Bearer {TOKEN}"
    return headers

def test_ebird(mock_requests):
    mock_requests.get(f"{BASE_URL}/ebird", json={"message": "ok"}, status_code=200)
    response = requests.get(f"{BASE_URL}/ebird", headers=get_headers())
    assert response.status_code in EXPECTED_STATUS_200_500

def test_xenocanto(mock_requests):
    mock_requests.get(f"{BASE_URL}/xenocanto", json={"message": "ok"}, status_code=200)
    response = requests.get(f"{BASE_URL}/xenocanto", headers=get_headers())
    assert response.status_code in EXPECTED_STATUS_200_500

def test_sensitivity(mock_requests):
    mock_requests.get(f"{BASE_URL}/sensitivity", json={"message": "ok"}, status_code=200)
    response = requests.get(f"{BASE_URL}/sensitivity", headers=get_headers())
    assert response.status_code in EXPECTED_STATUS_200_500

def test_databird(mock_requests):
    mock_requests.get(f"{BASE_URL}/dataBird", json={}, status_code=404)
    response = requests.get(f"{BASE_URL}/dataBird", headers=get_headers())
    assert response.status_code in EXPECTED_STATUS_200_500_404
```

Fig.1. Pruebas de API

3.2.2 Pruebas de Ingesta (Integración MQTT y BBDD)

Para asegurar la fiabilidad del flujo **Edge-to-Cloud**, se implementó un conjunto de pruebas de integración que simula el comportamiento de un dispositivo Jetson.

- Cliente MQTT Real: Se utiliza la librería paho.mqtt.client para instanciar un cliente MQTT real que se conecta al broker (Mosquitto) de la plataforma.
- Publicación de Datos Simulados: Las pruebas cargan datos de ejemplo (desde un test_data.json) y los publican en los tópicos MQTT correspondientes (ej. a4birds/cam/segment para detecciones y a4birds/cam/heatmap/metadata/CAM456 para mapas de calor).
- Verificación Asíncrona: Dado que la ingesta es asíncrona, la prueba espera un tiempo prudencial (time.sleep) para permitir que el ai4birds-ingest-service (suscrito a esos tópicos) reciba, procese y almacene el mensaje.

3.2.3 Validación en Base de Datos

- Tras la espera, la prueba se conecta directamente a la base de datos PostgreSQL (usando un singleton de conexión).
- Ejecuta consultas SQL (SELECT * FROM ...) para verificar que los datos publicados por MQTT han sido persistidos correctamente en las tablas (segment_data, heatmap_image, bird_statistics).
- Se comprueba que los datos en la BBDD coinciden con los datos enviados en el payload de MQTT.

3.2.4 Gestión del Entorno

Se utilizan pytest.fixture para inicializar el cliente MQTT y para **limpiar la base de datos** (cleanup_db) después de cada ejecución, asegurando que las pruebas sean atómicas y no interfieran entre sí.

```python
import pytest
import time
import json
import base64

from ai4birds_ingest_service.model.db import PostgresSingleton

def load_test_data():
    with open('test_data/test_data.json', 'r') as file:
        return json.load(file)

test_data = load_test_data()


@pytest.mark.parametrize("test_payload", test_data['segment_data'])
def test_segment_data(mqtt_client, test_payload):

    mqtt_client.publish("a4birds/cam/segment", json.dumps(test_payload))
    time.sleep(5)

    db = PostgresSingleton.getInstance()
    db.connect()
    db.execute("SELECT * FROM segment_data WHERE camera_id = %s ORDER BY id DESC LIMIT 1", ("CAM123",))
    row = db.fetchone()
    db.close()

    assert row is not None
    assert row[1] == "CAM123"  # camera
    assert row[2] == 1         # segment_idx


@pytest.mark.parametrize("test_payload", test_data['heatmap_data'])
def test_heatmap_data(mqtt_client, test_payload):

    iamge_path = "../heatmaps/heatmap.png"
    with open(iamge_path, 'rb') as image_file:
        image_data = image_file.read()

    mqtt_client.publish("a4birds/cam/heatmap/metadata/CAM456", json.dumps(test_payload))
    mqtt_client.publish("a4birds/cam/heatmap/image/CAM456", image_data)
    time.sleep(5)

    db = PostgresSingleton.getInstance()
    db.connect()
    db.execute("SELECT * FROM heatmap_image WHERE camera_id = %s ORDER BY id DESC LIMIT 1", ('CAM456',))
    row = db.fetchone()
    db.close()
```

Fig.2. Gestión de entrono

3.2.5 Pruebas de Interfaces y Flujos de Usuario

A continuación, se presenta una tabla con las pruebas funcionales realizadas para verificar el funcionamiento de las interfaces (UI/UX) y los flujos de usuario de la plataforma. Cada prueba incluye el estado de funcionamiento esperado y comentarios sobre la validación del flujo.

Prueba	Estado	Salida Esperada	Comentarios
Visualización del Mapa (Usuario no registrado)	FUNCIONAL	El usuario puede navegar por el mapa, ver cámaras públicas y las capas por defecto (Exclusión IDECYL, DataBird).	Se valida el acceso público a los datos de contexto y la correcta carga de las capas base.
Activación de Capas Interactivas (Mapa)	FUNCIONAL	El usuario puede activar las capas de Recurso Eólico y Cuadrícula Mesoescalar. Al hacer clic, se muestran los *popups* con las gráficas y los datos correspondientes.	Se valida la interactividad del mapa y la correcta invocación y visualización de datos de los *endpoints* (/windmap, etc.).
Proceso de Solicitud de Registro	FUNCIONAL	El usuario rellena el formulario de "Solicitar rol". La plataforma envía un correo de activación al administrador para su aprobación manual.	El flujo de registro mediatizado funciona. La cuenta no se crea hasta que el administrador la aprueba.
Proceso de Inicio de Sesión (Login)	FUNCIONAL	Un usuario registrado introduce sus credenciales. El backend emite un JWT, que el frontend almacena para futuras peticiones.	La autenticación mediante JWT es correcta. El usuario accede a las vistas y permisos de un usuario registrado.
Visualización de Cámaras (Usuario Registrado)	FUNCIONAL	La vista de "Cámaras" muestra tanto las cámaras públicas como las privadas del usuario. Aparece el botón "Añadir Cámara".	El filtrado de cámaras por propietario y el acceso a la funcionalidad de registro funcionan como se espera.
Registro de Nueva Cámara	FUNCIONAL	El usuario rellena el formulario modal (Nombre, URL, Coordenadas, Visibilidad). La cámara se registra en la BBDD y aparece en el listado.	El flujo de registro de nuevos dispositivos (cámaras) es completamente funcional.
Acceso a Detalle de Cámara	FUNCIONAL	El usuario accede a la vista de detalle y puede ver las estadísticas de especies, los datos en crudo de	Se valida que los usuarios registrados pueden acceder a las analíticas avanzadas de

Prueba	Estado	Salida Esperada	Comentarios
(Usuario Registrado)		detección y el *heatmap* más reciente.	las cámaras que tienen permiso para ver.
Acceso Restringido (Usuario no registrado)	FUNCIONAL	Un usuario no registrado que intenta acceder a la vista de detalle de una cámara (ej. desde el mapa) es redirigido o se le deniega el acceso a las estadísticas/heatmaps.	La capa de seguridad (ai4birds-coordinate-service) funciona correctamente, protegiendo los *endpoints* de datos sensibles.
Acceso al Panel de Administración	FUNCIONAL	Un usuario registrado (con rol de admin) accede al panel y puede visualizar el JSON en crudo de los *endpoints* de XenoCanto, eBird e IDECYL.	Se valida el acceso a la vista de depuración técnica para la supervisión de los conectores.
Acceso al Blog	FUNCIONAL	Cualquier usuario, registrado o no, puede acceder a la vista del Blog y leer los artículos publicados.	La sección de contenido público y divulgación es accesible para todos.

Tabla.1. Resultado de pruebas de interfaces y flujos de usuarios

Los resultados de las pruebas demuestran que todas las funcionalidades críticas del sistema IA4Birds se encuentran operativas y validadas, cumpliendo con los criterios de aceptación definidos en la fase de diseño y con las especificaciones de la arquitectura modular.

El módulo de visualización geoespacial mostró una respuesta fluida y estable, validando la correcta integración entre la interfaz de usuario y los servicios API (coordinate e ingest). Las funciones de autenticación y seguridad basadas en JWT presentaron una robustez satisfactoria frente a intentos de acceso no autorizado, garantizando la segregación de roles y la protección de datos sensibles.

El flujo de gestión de cámaras IoT evidenció una alta estabilidad en la interacción entre el frontend y la base de datos, con tiempos de respuesta adecuados y una correcta actualización del estado de los dispositivos en tiempo real.

Finalmente, la validación de los componentes públicos (como el blog y las capas del mapa) confirma el cumplimiento del objetivo de comunicación y divulgación del proyecto, asegurando el acceso abierto a la información relevante para la sociedad, investigadores y gestores ambientales.

3.2.6 Instalación prototipo

Para la instalación del prototipo se han barajado diferentes opciones. El aspecto más relevante a la hora de seleccionar la ubicación ideal pasa por que en dicha ubicación existan diferentes especies de aves que se puedan grabar y catalogar. La instalación prototipo se encuentra en AIR Institute, que cuenta con un edificio de investigación que se encuentra en un punto cercano a Las Dunas, u centro de recuperación de aves y en una zona en altura privilegiada donde se

observan varias especies de alto interés como pueden ser cigüeñas, milanos, cernícalos y otras especies de paso que han servido para testear la solución de una manera ágil, permitiendo cambios en el sistema y pudiendo entrenar los modelos de IA con especies de la zona, siendo más importante en este aspecto el tener cerca la herramienta para posibles mejoras o reparaciones que el disponer de la solución en zonas alejadas u otras zonas de protección de las aves, que pudieran haber sido buena solución para una mayor variabilidad de las aves pero que tenía puntos en contra que pesaban más que los beneficios.

A continuación se muestra unas imágenes de la instalación prototipo:

Fig.4. Instalación prototipo

4 Conclusión

La arquitectura híbrida Edge + Cloud ha demostrado ser una solución eficiente para gestionar grandes volúmenes de datos en tiempo real, reduciendo la latencia y optimizando los recursos de red. La separación de responsabilidades entre los componentes —Ingest Service, Coordinate Service, Backend y Frontend—, junto con la contenedorización mediante Docker y la automatización del despliegue a través de GitHub Actions, garantiza una estructura modular, escalable y fácilmente mantenible. Esta aproximación basada en microservicios ha permitido alcanzar altos niveles de rendimiento, resiliencia y trazabilidad, elementos esenciales para el funcionamiento continuo de un sistema de monitorización ambiental distribuido.

Las pruebas realizadas —unitarias, de integración, E2E y funcionales— confirman la estabilidad del sistema y la coherencia de sus flujos de información, desde la detección de aves en el Edge hasta su representación visual en el mapa geoespacial. Los resultados validan la correcta interoperabilidad entre los componentes, la seguridad en la transmisión de datos mediante TLS y JWT, y la accesibilidad de las interfaces desarrolladas, tanto para usuarios técnicos como para el público general.

En su conjunto, este entregable y los E4.1 y E4.2 demuestran que IA4Birds cumple con los objetivos planteados en la **A4. Sistema de gestión de la información de aves y tecnologías de visualización**,