Te adjunto mi proyecto EAUTPL Para que lo analices completamente Que está diseñado Para una finca bovina productora de leche. Y Deseo. Que se cumplan los principios SOLID y de arquitectura limpia para garantizar un sistema modular, mantenible y escalable.

los problemas de repetición de funciones y de base de datos y mejora la coherencia, mantenibilidad y escalabilidad del sistema.

Qué necesito que hagas:

Reorganizar la estructura del proyecto para reflejar las capas de la Arquitectura Limpia:

1. \*\*Eliminación de redundancias en módulos:\*\*

- centralizar las funciones comunes en un módulo de "Entidades" y "Servicios Compartidos".

- eliminaro la duplicación de lógica de negocio entre módulos.

2. \*\*Normalización numero 4 de la base de datos :\*\*

- crear un esquema unificado que elimina redundancias.

- implementar relaciones adecuadas entre tablas para mantener la integridad referencial.

3. Arquitectura Limpia y SOLID:

* Separación de Concerns: Aplica los principios SOLID para crear un código modular, mantenible y escalable.
* Capas: Define claramente las capas de tu aplicación (presentación, negocio, datos) y establece una comunicación clara entre ellas.
* Dependencias: Asegúrate de que las dependencias fluyan en una sola dirección y que los módulos de bajo nivel no dependan de los de alto nivel.

4. \*\*Implementación de patrones de diseño:\*\*

- Se utiliza el patrón Repository para abstraer el acceso a datos.

- Se implementa el patrón Factory para la creación de objetos complejos.

- Se aplica el patrón Observer para manejar eventos y actualizaciones en tiempo real.

5. Identificación de Microservicios:

* Dominio del Problema: Analiza a fondo el sistema para identificar las funcionalidades que pueden ser descompuestas en servicios independientes. Por ejemplo, un sistema
* Coherencia y Autonomía: Cada microservicio debe tener una responsabilidad bien definida y poder ser desarrollado, desplegado y escalado de forma independiente.

6. Selección de Patrones de Diseño:

* RESTful API: Ideal para exponer la funcionalidad de los microservicios a través de HTTP.
* CQRS: Perfecto para separar las operaciones de lectura y escritura, mejorando el rendimiento y la escalabilidad.
* Event Sourcing: Permite capturar el historial de cambios, facilitando la auditoría y la recuperación de datos.
* Circuit Breaker: Evita la propagación de fallas y protege el sistema de sobrecargas.
* Otros: Considera patrones como Saga, API Gateway y Bulkhead, según las necesidades específicas de tu sistema.

7. Inteligencia Artificial (IA):

* Microservicios de IA: Crea microservicios específicos para tareas de IA, como procesamiento de lenguaje natural, visión por computadora, etc.
* Frameworks de IA: Utiliza frameworks como TensorFlow, PyTorch o scikit-learn para desarrollar tus modelos de IA.
* Integración: Integra los microservicios de IA con el resto del sistema utilizando APIs RESTful o gRPC.

8. Herramientas y Tecnologías:

* Frameworks Python: Flask, Django REST framework para construir APIs RESTful.
* Orquestación: Docker Compose o Kubernetes para gestionar el despliegue y escalado de los microservicios.
* Mensajería: RabbitMQ, Kafka para la comunicación asíncrona entre microservicios.
* Bases de Datos: PostgreSQL, MongoDB, Cassandra, según las necesidades de cada microservicio.
* Cloud: Plataformas como AWS, GCP o Azure para desplegar y escalar tu sistema de forma eficiente.

9. Pruebas y Monitoreo:

* Pruebas Unitarias: Asegúrate de que cada componente funcione correctamente de forma aislada.
* Pruebas de Integración: Verifica que los microservicios se comuniquen correctamente entre sí.
* Pruebas de Rendimiento: Evalúa la capacidad del sistema para manejar cargas altas.
* Monitoreo: Utiliza herramientas como Prometheus y Grafana para monitorear el rendimiento y la salud de tus microservicios.