UNIVERSIDAD POPULAR DEL CESAR
FACULTAD DE INGENIERÍAS Y TECNOLÓGICAS
ESPECIALIZACIÓN EN INGENIERÍA DE SOFTWARE

TALLER No. 2
Diseño de solución con patrones de diseño
Soluciones agro (Software de gestión de procesos de actividad piscícola)

INTEGRANTES GRUPO 06:
Marryy Selena Garay Larios
Abraham Sarabia Sereno

DOCENTE: Ing. Jairo Francisco Seoanes León
ASIGNATURA: Patrones de Diseño de Software
Junio de 2025 - Periodo 2025-I

Actividad 1: Análisis del escenario y definición de requerimientos

Descripción del caso de estudio:

El sector piscícola colombiano enfrenta dificultades para gestionar de forma eficiente los procesos de producción de peces, especialmente en el levante y engorde de tilapia. Las tareas como el control de estanques, el monitoreo de la población piscícola, el registro de mortalidad, el consumo de alimentos y la planificación estructural se hacen manualmente, generando errores, demoras y baja trazabilidad.

Soluciones Agro busca automatizar la gestión de estos procesos en una plataforma web que integre:

- Módulos de monitoreo productivo
- Gestión de inventarios y alimentación
- Reportes financieros y proyecciones
- Alertas automáticas y recomendaciones

Requerimientos funcionales:

Identificador	Descripción	Estado	Prioridad
RF01	El sistema deberá permitir registrar, eliminar, actualizar y listar la información de cada estanque con su información (dimensiones, ubicación, etc.).	Alta	Alta
RF02	El sistema deberá permitir registrar, eliminar, actualizar y listar las características del cultivo incluyendo (peso, edad, etapa, cantidad, etc.).	Alta	Alta
RF03	El sistema deberá permitir registrar, eliminar, actualizar y listar entradas y salidas de alimentos.	Alta	Alta
RF04	El sistema deberá permitir generar proyecciones automáticas de consumo según la población y su crecimiento.	Media	Media
RF05	El sistema deberá permitir registrar y monitorear la población de peces por cada estanque (crecimiento y desarrollo del pez).	Alta	Alta
RF06	El sistema deberá permitir generar alertas en caso de bajas o alteraciones significativas en la población (bajo inventario, mortalidad alta, etc.).	Alta	Alta
RF07	El sistema deberá permitir registrar, eliminar, actualizar y listar eventos como enfermedades, mortalidad o condiciones climáticas.	Media	Media

RF08	El sistema deberá permitir generar informes de crecimiento, reproducción y mortalidad por estanques.	Alta	Alta
RF09	El sistema deberá permitir planificar la estructura de los estanques (capacidad, distribución, condiciones).	Media	Baja
RF10	El sistema deberá permitir generar informes consolidados sobre el desempeño productivo general.	Alta	Alta
RF11	El sistema deberá permitir configurar parámetros clave: unidades de medida, moneda.	Media	Media
RF12	El sistema deberá permitir consultar históricos de producción, consumo y eventos registrados.	Alta	Alta
RF13	El sistema deberá permitir generar reportes de costos y gastos derivados de la gestión productiva.	Media	Media
RF14	El sistema deberá permitir consultar recomendaciones o guías para optimizar procesos piscícolas.	Baja	Baja
RF15	El sistema deberá permitir acceso al sistema desde dispositivos móviles y de escritorio.	Alta	Alta
RF16	El sistema deberá permitir exportar datos clave a Excel o PDF.	Media	Media
RF17	El sistema deberá permitir registrar, eliminar, actualizar y listar proveedores de alimentos, insumos y equipos.	Media	Baja
RF18	El sistema deberá permitir a los usuarios acceder al sistema mediante inicio de sesión con credenciales seguras.	Alta	Alta
RF19	El sistema deberá permitir registrar, consultar, modificar y eliminar información de usuarios registrados, incluyendo sus roles: administradores y operarios.	Alta	Alta
RF20	El sistema deberá permitir registrar, eliminar, actualizar y listar las ventas realizadas y consultar historiales.	Alta	Alta
RF21	El sistema deberá permitir registrar, eliminar, actualizar y listar el consumo diario de alimentos por estanques.	Media	Media

Requerimientos no funcionales:

Identificador	Descripción	Estado	Prioridad
RNF01	El sistema debe estar disponible las 24 horas del día, los 7 días de la semana, 365 días al año.	Alta	Alta
RNF02	Toda funcionalidad o transacción debe responder en menos de 15 segundos.	Alta	Alta
RNF03	El sistema no debe presentar inactividad no planificada superior a 10 horas por bimestre o 40 horas al año.	Alta	Alta
RNF04	El sistema debe permitir aumentar la capacidad para ofrecer servicios a más usuarios sin degradar el rendimiento.	Alta	Alta

RNF05	El sistema debe ser 100% web y permitir su administración y parametrización desde un navegador.	Alta	Alta
RNF06	El sistema debe proporcionar interfaces intuitivas y fáciles de usar para los usuarios.	Alta	Alta
RNF07	Debe contar con manuales de usuario claros y bien estructurados.	Alta	Media
RNF08	Debe garantizar la correcta visualización en PC, tabletas y teléfonos inteligentes.	Alta	Alta
RNF11	El sistema debe permitir que todas las funcionalidades sean accesibles mediante el teclado.	Alta	Media
RNF12	El sistema debe cumplir con las políticas de seguridad de Soluciones Agro.	Alta	Alta
RNF13	Debe registrar todas las acciones relevantes de usuarios y administradores en un log de auditoría.	Alta	Alta
RNF14	Debe permitir la integración con sistemas externos (contables o productivos).	Media	Media
RNF15	Debe facilitar el mantenimiento correctivo y evolutivo.	Media	Media
RNF16	El sistema debe permitir crear réplicas para escalar horizontalmente.	Alta	Alta
RNF17	Cumplir con normativas vigentes de protección de datos y agroindustria.	Alta	Alta
RNF18	El sistema debe superar pruebas de carga (mínimo 1000 usuarios concurrentes) y seguridad.	Alta	Alta

Actividad 2: Diseño de la arquitectura de la solución.

Tipo de arquitectura seleccionada:

Arquitectura Limpia (Clean Architecture)

Justificación:

- Separa capas de negocio, presentación e infraestructura
- Facilita pruebas, mantenimiento y escalabilidad
- Permite implementar patrones de diseño fácilmente

Aplicación de patrones de diseño:

Patrones Creacionales:

Patrón	Aplicación
Singleton	Servicios de configuración, logging, conexión a BD

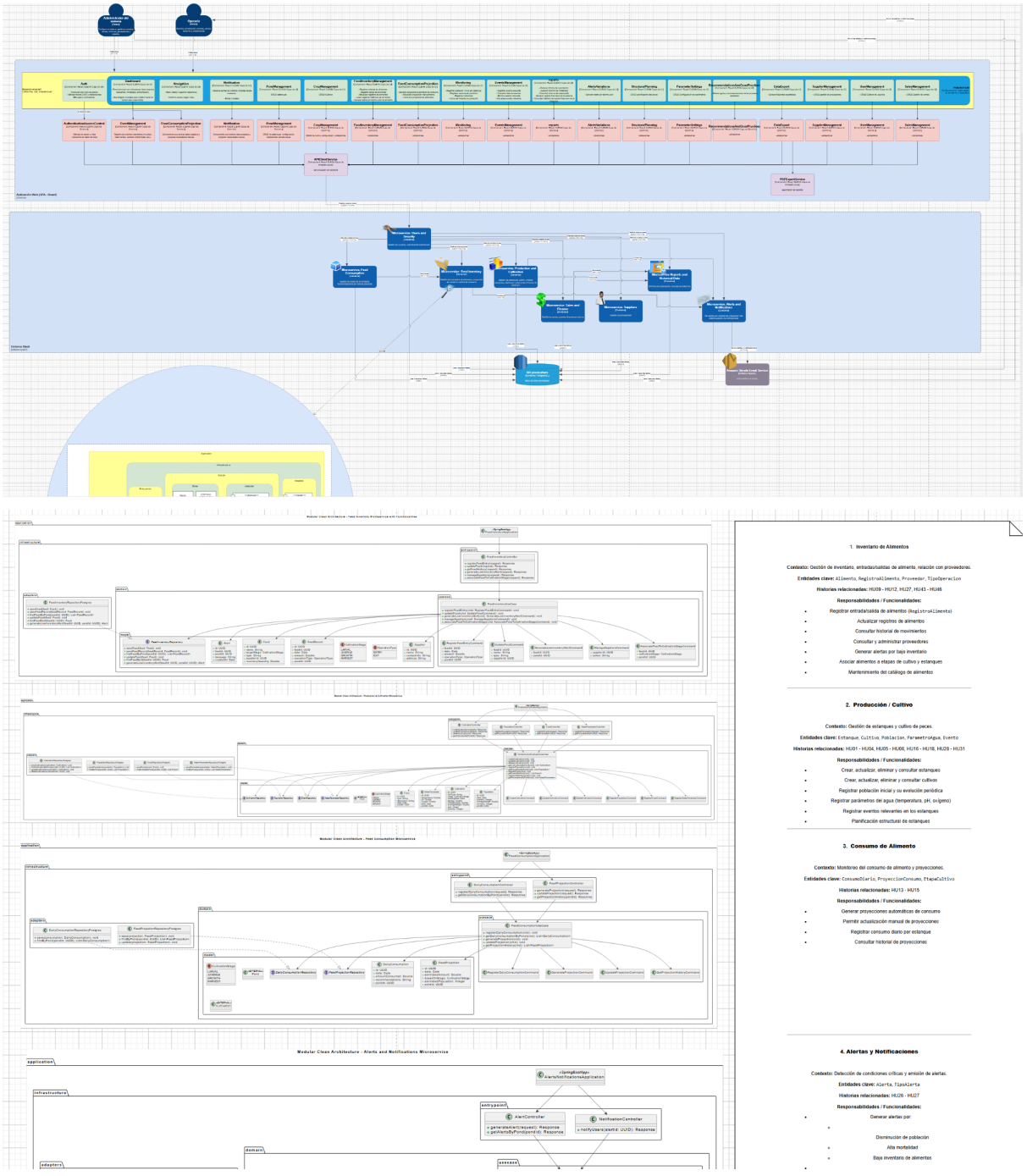
Factory Method	Creación de reportes financieros e informes
Builder	Objetos complejos como Reporte, Usuario o Parámetros del sistema
Prototype	Clonado de estructuras para informes personalizados
Abstract Factory	Generación de interfaces responsivas según tipo de dispositivo

Patrones Estructurales:

Patrón	Aplicación
Adapter	Conexión con sistemas contables externos
Facade	Unificar generación de reportes con múltiples fuentes de datos
Composite	Tarjetas del panel de indicadores agrupadas en jerarquía visual
Decorator	Añadir funciones extra a reportes sin modificar su lógica central
Proxy	Validación de permisos antes de ejecutar acciones sensibles (roles)
Bridge	Desacoplar abstracción del reporte y su formato de salida (PDF, Excel)

Diagrama de arquitectura y clases:

A continuación se agrega un link que dirige al archivo C-4Software de gestión de procesos de actividad piscícola para su mejor visualización.



link:
https://drive.google.com/file/d/1FDIOJQ3sCqoi0yLa3s664iEdS2BfrHRo/view?usp=sharing

Actividad 3: Implementación de la solución

Stack tecnológico:

Componente	Herramienta	Justificación
Frontend	React con Tailwind	Interfaz responsive
Backend	Spring Boot (Java)	Inyección de dependencias, REST
Base de datos	PostgreSQL	Relacional, escalable
Seguridad	Spring Security + JWT	Autenticación/roles segura
Reportes	JasperReports o PDFKit	Generación de informes
Comunicación	REST API (HTTP/HTTPS)	Estándar y seguro

Integración de patrones con el framework:

Spring Boot:

- `@Component`, `@Service`, `@Autowired` para Singleton y Factory
- Controladores REST como fachada (`@RestController`)
- `@Configuration` para objetos compartidos (config/global)
- Bean personalizado para Decorator de servicios (reporte/alertas)

React:

- Componentes reutilizables → Composite
- Hooks → Decorator-like funcionalidad
- Context API → Singleton compartido (sesión, usuario)

Explicación por patrón (ejemplo):

- Factory Method: Se usa una clase `ReporteFactory` que genera diferentes tipos de informes según los parámetros (población, inventarios, ventas).
- Decorator: Se agregan decoradores al generador de informes para incluir estadísticas opcionales como tendencias históricas.

- Facade: Una clase `GeneradorDeReportesFacade` agrupa múltiples servicios: `CultivoService`, `VentasService`, `PoblacionService` para producir un informe consolidado.

Conclusiones

El uso de patrones de diseño permite desarrollar software robusto, escalable y mantenible.Gracias a la aplicación de patrones creacionales (como Singleton, Factory Method y Builder) y estructurales (como Facade, Adapter y Composite), fue posible diseñar una solución modular que facilita la evolución del sistema a futuro.

La arquitectura limpia (Clean Architecture) promueve una separación clara de responsabilidades.
Esto permitió dividir el sistema en capas independientes (dominio, aplicación, infraestructura y presentación), facilitando pruebas unitarias, mantenimiento y la incorporación de nuevas funcionalidades sin afectar el núcleo del sistema.

La integración con frameworks modernos como Spring Boot y React mejora significativamente el desarrollo.
El uso de herramientas con soporte nativo para inyección de dependencias, configuración modular y generación automática de componentes facilitó la implementación de los patrones definidos.

La solución propuesta responde de manera efectiva a las necesidades reales del sector piscícola.
Al abordar problemáticas concretas como el control de inventarios, monitoreo de población y reportes financieros, Soluciones Agro se perfila como una herramienta útil y práctica para optimizar la gestión acuícola.

Lecciones Aprendidas

Diseñar con patrones desde el inicio evita sobrecostos en el mantenimiento.
Incluir patrones como Builder y Factory Method desde la fase de diseño ayudó a crear una base de código flexible y reutilizable.

El análisis previo de los requisitos funcionales y no funcionales es esencial para seleccionar la arquitectura adecuada.
Identificar correctamente los módulos críticos (como alimentación, población y reportes) facilitó la definición de los puntos de extensión del sistema y la aplicación de patrones.

Los patrones estructurales simplifican la complejidad de los módulos más críticos.
Patrones como Facade y Adapter permitieron abstraer lógicas complejas y facilitar la comunicación con servicios externos o diferentes capas del sistema.

La documentación técnica clara es tan importante como el código.
Durante el desarrollo del proyecto se evidenció la necesidad de mantener actualizados los diagramas, descripciones de casos de uso, historias de usuario y trazabilidad para facilitar el trabajo en equipo y garantizar calidad.

Los principios SOLID y GRASP deben ir de la mano con los patrones.
La aplicación de estos principios junto a los patrones de diseño garantiza un diseño más orientado al cambio, bajo acoplamiento y alta cohesión.

Referencias Bibliográficas

1. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

2. Spring Framework Documentation. (2024). https://docs.spring.io/spring-framework/docs/current/reference/html/

3. Freeman, E., Freeman, E., Sierra, K., & Bates, B. (2004). *Head First Design Patterns*. O'Reilly Media.

4. Martin, R. C. (2018). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.