Sistema de Configuración de Planes de Suscripción para una Empresa de Streaming

Integrantes:
Sergio Alberto Ariza Franco
James David Vanstrahlen Molina
Bladimir Jesus Junior Vargas Rios
Nemer Jose Velandia Soto

Universidad Popular del Cesar
Especialización Ingeniería de Software
Patrones Diseño de Software
Jairo Francisco Sehoanes Leon
2025-1

Problema

Una empresa de streaming que ofrece contenido de video necesita un sistema para gestionar sus planes de suscripción. Los clientes pueden elegir entre diferentes tipos de planes (básico, estándar, premium), que varían según la calidad de video, la cantidad de dispositivos en los que se puede ver contenido y si incluye anuncios o no.

El sistema también debe permitir la personalización de los planes agregando servicios adicionales, como contenido exclusivo o almacenamiento extra para descargas. Además, algunos usuarios pueden querer clonar configuraciones de suscripciones anteriores para reutilizar sus preferencias de plan y servicios adicionales.

Patrones Creacionales Aplicados:

- **Factory Method**: Para crear los diferentes tipos de planes de suscripción (básico, estándar, premium).
- **Builder**: Para permitir la personalización del plan de suscripción, agregando servicios adicionales como contenido exclusivo o almacenamiento extra.
- **Prototype**: Para clonar configuraciones de suscripción anteriores y ajustarlas según las necesidades actuales del cliente.
- **Strategy:** Para aplicar diferentes estrategias de cálculo del precio final del plan, dependiendo del contexto, como promociones, descuentos por región o condiciones especiales del usuario.

Justificación de los Patrones Seleccionados

- **Factory Method:** Se utilizó para permitir la creación polimórfica de planes de suscripción (básico, estándar, premium), desacoplando la lógica de instanciación del cliente.
- **Builder:** Se aplicó para facilitar la construcción de planes complejos mediante encadenamiento de métodos, mejorando la legibilidad y evitando constructores extensos.
- **Prototype:** Permite clonar configuraciones existentes y ajustarlas, brindando una forma eficiente de reutilizar suscripciones anteriores.
- **Strategy:** Se integró para implementar distintas estrategias de cálculo del precio final sin modificar la lógica del objeto PlanSuscripcion, facilitando su extensión dinámica.

Solución:

- **Factory Method** gestionará la creación de los planes de suscripción según el tipo de servicio elegido por el usuario (básico, estándar, premium).
- **Builder** permitirá personalizar el plan agregando servicios adicionales y ajustando las preferencias de calidad y dispositivos.
- **Prototype** permitirá a los usuarios clonar configuraciones anteriores y ajustar detalles menores en sus suscripciones futuras.
- **Strategy** permitirá aplicar dinámicamente distintas formas de calcular el precio del plan, de acuerdo con políticas comerciales o promociones activas, sin modificar la lógica del objeto `PlanSuscripcion`.

Requerimientos Funcionales

id	1
Nombre	Configuración y Personalización de Planes de Suscripción
Descripción	El sistema debe permitir que los usuarios configuren su plan de suscripción eligiendo entre opciones como tipo de plan (básico, estándar, premium), calidad de video, cantidad de dispositivos, inclusión o exclusión de anuncios, y servicios adicionales como contenido exclusivo o almacenamiento extra. La configuración debe reflejarse en tiempo real en el resumen de precio del plan.
importancia	Alta
prioridad	Alta

id	2
Nombre	Clonación de Configuraciones Anteriores de Plan
Descripción	El sistema debe permitir a los usuarios clonar una configuración previa de suscripción y editarla antes de finalizar el proceso. Esto facilitará la reutilización de preferencias anteriores sin necesidad de configurar el plan desde cero.
importancia	Media
prioridad	Alta

id	3
Nombre	Aplicación de Estrategias de Cálculo de Precio
Descripción	El sistema debe aplicar diferentes estrategias de cálculo del precio final de un plan de suscripción, como precios base, promociones para estudiantes o ajustes por región. Estas estrategias deben ser intercambiables sin alterar la lógica del plan.
importancia	Alta
prioridad	Media

id	4
Nombre	Selección del Tipo de Plan de Suscripción
Descripción	El sistema debe permitir que el usuario seleccione entre los planes predefinidos: Básico, Estándar y Premium. Cada tipo de plan incluye una configuración por defecto en cuanto a calidad de video, dispositivos permitidos y anuncios.
importancia	Alta
prioridad	Alta

id	5
Nombre	Gestión de Servicios Adicionales
Descripción	El sistema debe permitir a los usuarios agregar o quitar servicios adicionales como almacenamiento extra o contenido exclusivo, modificando el precio total del plan en tiempo real.
importancia	Alta
prioridad	Alta

id	6
Nombre	Visualización del Precio Total del Plan
Descripción	El sistema debe mostrar en todo momento el precio acumulado del plan mientras el usuario realiza modificaciones en los atributos del plan o añadir servicios adicionales.
importancia	Alta
prioridad	Alta

id	7
Nombre	Restaurar configuración previa
Descripción	El sistema debe ofrecer la opción de restaurar configuraciones de planes anteriores previamente guardados por el usuario, cargando automáticamente los valores usados anteriormente.
importancia	Media

prioridad	Media

id	8
Nombre	Vista previa del plan configurado
Descripción	Antes de confirmar la suscripción, el sistema debe mostrar un resumen detallado del plan: tipo, calidad, dispositivos, anuncios, servicios extra y precio final.
importancia	Alta
prioridad	Alta

id	9
Nombre	Confirmación de suscripción
Descripción	El sistema debe permitir al usuario confirmar la suscripción una vez completada la configuración y proceder al proceso de pago o validación.
importancia	Alta
prioridad	Alta

id	10
Nombre	Aplicación de estrategia de cálculo de precio
Descripción	El sistema debe aplicar una estrategia de cálculo según condiciones específicas (promoción activa, región del usuario, condición estudiantil, etc.)

	sin que el usuario lo configure manualmente.
importancia	Alta
prioridad	Media

Requerimientos No Funcionales

RNF	RNF 1
Nombre	Escalabilidad dinámica del sistema de suscripciones
Descripción	El sistema debe ser capaz de soportar al menos 5,000 usuarios concurrentes gestionando configuraciones y personalizaciones de planes sin afectaciones en el rendimiento ni en la generación del resumen de precio.
Clasificación	Escalabilidad
Criterios de aceptación	Pruebas de carga: El sistema debe simular hasta 5,000 usuarios personalizando planes al mismo tiempo, y mantener un tiempo de respuesta menor a 3 segundos en el proceso de construcción del plan y cálculo de precio. Monitoreo de rendimiento: Se debe implementar monitoreo continuo para asegurar que no haya degradaciones en el rendimiento del motor de configuración y precios.

RNF	RNF 2
Nombre	Flexibilidad para extensión de nuevos planes y estrategias
Descripción	El sistema debe permitir la adición de nuevos tipos de planes o estrategias de precios sin modificar la lógica existente, apoyándose en el uso de patrones de diseño como Factory Method y Strategy.
Clasificación	Mantenibilidad / Extensibilidad
Criterios de aceptación	Evolución modular: La incorporación de un nuevo plan o una nueva estrategia de precio debe requerir solo la creación de nuevas clases, sin modificaciones en las existentes. Validación de regresión: La integración de nuevas estrategias no debe romper funcionalidades actuales del sistema, validado por pruebas unitarias automatizadas.

RNF	RNF 3
Nombre	Seguridad en la Configuración y Gestión
Descripción	El sistema debe garantizar que solo usuarios autenticados puedan acceder al módulo de configuración de planes, y que las acciones estén protegidas contra ataques como inyección, CSRF o acceso no autorizado.
Clasificación	Seguridad

Criterios de aceptación	Autenticación y autorización: Validar que solo usuarios autenticados puedan crear, editar o clonar planes. Validaciones de entrada: Aplicar validaciones server-side para prevenir ataques como XSS o inyecciones. Protección de datos: La información personal del usuario asociada al plan debe estar cifrada y protegida.

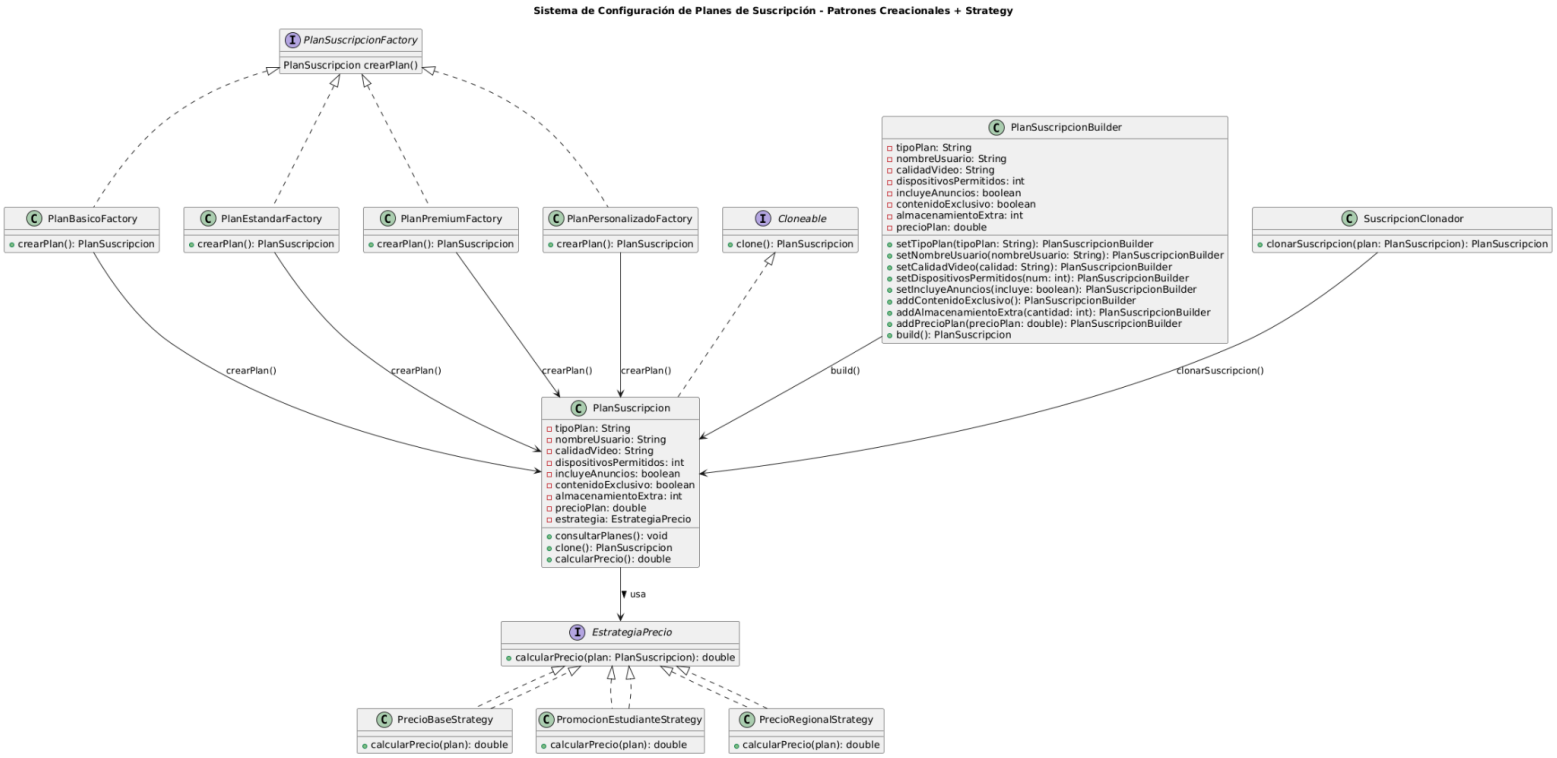
RNF	RNF 4
Nombre	Usabilidad de la Interfaz de Configuración
Descripción	La interfaz de configuración de planes debe ser intuitiva, permitiendo al usuario completar el proceso sin necesidad de documentación externa.
Clasificación	Usabilidad
Criterios de aceptación	Tiempos de interacción: El tiempo medio para configurar un plan completo no debe superar los 2 minutos. Pruebas de usuario: Al menos el 90% de los usuarios debe poder completar el proceso de personalización sin asistencia en pruebas de usabilidad.

RNF	RNF 5
Nombre	Integración continua y despliegue automatizado

Descripción	El sistema debe estar preparado para integrarse en una pipeline CI/CD que permita automatizar pruebas y despliegues.
Clasificación	Mantenibilidad / DevOps
Criterios de aceptación	Pipeline: El sistema debe estar integrado con una pipeline (GitHub Actions, GitLab CI, etc.) que realice pruebas unitarias y despliegue a un entorno staging automáticamente. Rollback: En caso de error, debe permitir revertir a la versión anterior sin pérdida de datos ni interrupción del servicio.

RNF	RNF 6
Nombre	Monitorización del comportamiento del sistema
Descripción	El sistema debe contar con mecanismos de observabilidad (logs, métricas y trazas) para rastrear errores, latencias y uso de funcionalidades.
Clasificación	Observabilidad / Operación
Criterios de aceptación	Logs centralizados: Todas las operaciones críticas deben ser registradas con trazabilidad del usuario. Métricas de rendimiento: Se deben generar métricas sobre tiempo promedio de configuración de plan, errores por servicio, y uso de estrategias de precio.

Diagrama de clases



@startuml

title Sistema de Configuración de Planes de Suscripción - Patrones Creacionales + Strategy

' ====== Factory Method Pattern ======

interface PlanSuscripcionFactory {

    PlanSuscripcion crearPlan()

}

class PlanBasicoFactory implements PlanSuscripcionFactory {

    + crearPlan(): PlanSuscripcion

}

class PlanEstandarFactory implements PlanSuscripcionFactory {

    + crearPlan(): PlanSuscripcion

}

class PlanPremiumFactory implements PlanSuscripcionFactory {

    + crearPlan(): PlanSuscripcion

}

class PlanPersonalizadoFactory implements PlanSuscripcionFactory {

    + crearPlan(): PlanSuscripcion

}

' ====== Clase PlanSuscripcion ======

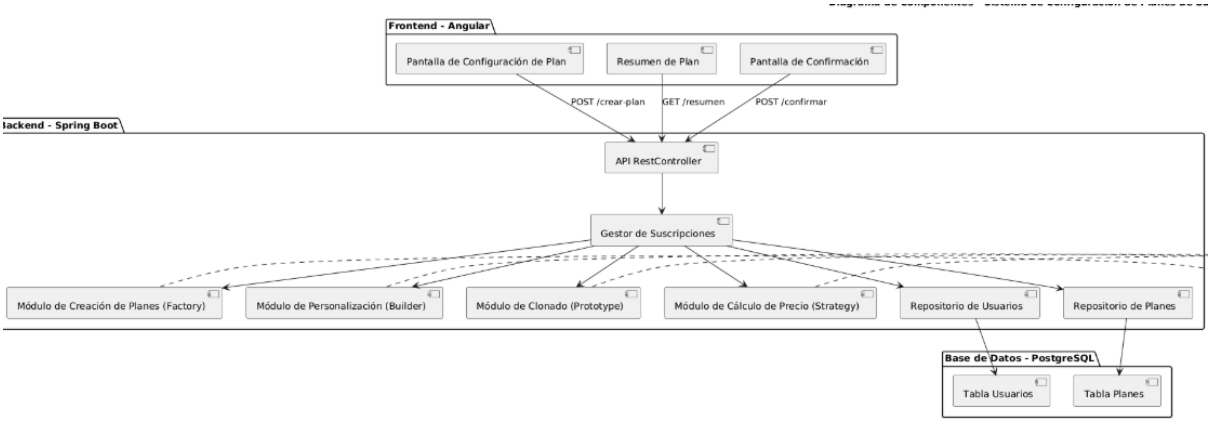
```
class PlanSuscripcion implements Cloneable {
    - tipoPlan: String
    - nombreUsuario: String
    - calidadVideo: String
    - dispositivosPermitidos: int
    - incluyeAnuncios: boolean
    - contenidoExclusivo: boolean
    - almacenamientoExtra: int
    - precioPlan: double
    - estrategia: EstrategiaPrecio
    + consultarPlanes(): void
    + clone(): PlanSuscripcion
    + calcularPrecio(): double
}

interface Cloneable {
    + clone(): PlanSuscripcion
}

' ====== Builder Pattern ======
class PlanSuscripcionBuilder {
    - tipoPlan: String
    - nombreUsuario: String
```

```
    - calidadVideo: String
    - dispositivosPermitidos: int
    - incluyeAnuncios: boolean
    - contenidoExclusivo: boolean
    - almacenamientoExtra: int
    - precioPlan: double

    + setTipoPlan(tipoPlan: String): PlanSuscripcionBuilder
    + setNombreUsuario(nombreUsuario: String): PlanSuscripcionBuilder
    + setCalidadVideo(calidad: String): PlanSuscripcionBuilder
    + setDispositivosPermitidos(num: int): PlanSuscripcionBuilder
    + setIncluyeAnuncios(incluye: boolean): PlanSuscripcionBuilder
    + addContenidoExclusivo(): PlanSuscripcionBuilder
    + addAlmacenamientoExtra(cantidad: int): PlanSuscripcionBuilder
    + addPrecioPlan(precioPlan: double): PlanSuscripcionBuilder
    + build(): PlanSuscripcion
}

' ====== Prototype Pattern ======
class SuscripcionClonador {
    + clonarSuscripcion(plan: PlanSuscripcion): PlanSuscripcion
}
```

```
' ====== Strategy Pattern ======
interface EstrategiaPrecio {
    + calcularPrecio(plan: PlanSuscripcion): double
}

class PrecioBaseStrategy implements EstrategiaPrecio {
    + calcularPrecio(plan): double
}

class PromocionEstudianteStrategy implements EstrategiaPrecio {
    + calcularPrecio(plan): double
}

class PrecioRegionalStrategy implements EstrategiaPrecio {
    + calcularPrecio(plan): double
}


' ====== Relaciones con cardinalidades y conexiones ======

' Builder construye PlanSuscripcion (1 a 1)
PlanSuscripcionBuilder --> PlanSuscripcion : build()
```

```
' Clonador clona PlanSuscripcion (1 a 1)

SuscripcionClonador --> PlanSuscripcion : clonarSuscripcion()


' Cada Factory produce un PlanSuscripcion (1 a 1)

PlanBasicoFactory --> PlanSuscripcion : crearPlan()

PlanEstandarFactory --> PlanSuscripcion : crearPlan()

PlanPremiumFactory --> PlanSuscripcion : crearPlan()

PlanPersonalizadoFactory --> PlanSuscripcion : crearPlan()


' Strategy (1 PlanSuscripcion usa 1 EstrategiaPrecio)

PlanSuscripcion --> EstrategiaPrecio : usa >


' Estrategias concretas implementan la interfaz

PrecioBaseStrategy ..|> EstrategiaPrecio

PromocionEstudianteStrategy ..|> EstrategiaPrecio

PrecioRegionalStrategy ..|> EstrategiaPrecio


@enduml
```

Diagrama de Componentes



@startuml

title Diagrama de Componentes - Sistema de Configuración de Planes de Suscripción

' COMPONENTES PRINCIPALES

package "Backend - Spring Boot" {

[API RestController] --> [Gestor de Suscripciones]

[Gestor de Suscripciones] --> [Módulo de Creación de Planes (Factory)]

[Gestor de Suscripciones] --> [Módulo de Personalización (Builder)]

[Gestor de Suscripciones] --> [Módulo de Clonado (Prototype)]

[Gestor de Suscripciones] --> [Módulo de Cálculo de Precio (Strategy)]

[Gestor de Suscripciones] --> [Repositorio de Usuarios]

[Gestor de Suscripciones] --> [Repositorio de Planes]

}

package "Frontend - Angular" {

```
    [Pantalla de Configuración de Plan]
    [Resumen de Plan]
    [Pantalla de Confirmación]

    [Pantalla de Configuración de Plan] --> [API RestController] : POST /crear-plan
    [Resumen de Plan] --> [API RestController] : GET /resumen
    [Pantalla de Confirmación] --> [API RestController] : POST /confirmar
}

package "Base de Datos - PostgreSQL" {
    [Repositorio de Usuarios] --> [Tabla Usuarios]
    [Repositorio de Planes] --> [Tabla Planes]
}

' LEGENDAS
note right of [Módulo de Creación de Planes (Factory)]
    Aplica Factory Method para instanciar
    planes según el tipo: básico, estándar, premium
end note

note right of [Módulo de Personalización (Builder)]
    Encapsula la lógica de construcción
    personalizada del plan
```

```
end note

note right of [Módulo de Clonado (Prototype)]
    Permite clonar un plan anterior y modificarlo
end note

note right of [Módulo de Cálculo de Precio (Strategy)]
    Aplica estrategias dinámicas de precio:
    base, promociones, descuentos regionales
end note

@enduml
```

Explicación de la Lógica:

- **Calidad de Video**:
  - **SD**: $5
  - **HD**: $6
  - **4K**: $7
- **Dispositivos Permitidos**:
  - Cada dispositivo adicional después del primero cuesta $1
- **Anuncios**:
  - Si el usuario no quiere anuncios, se suma $3.
- **Contenido Exclusivo**:
  - Si el usuario quiere contenido exclusivo, se suma $4.
- **Almacenamiento Extra**:
  - Cada GB de almacenamiento adicional cuesta $0.1

Stack de herramientas seleccionado

- Lenguaje: Java
- Framework Backend: Spring Boot
- Frontend: Angular
- Gestor de dependencias: Maven
- Base de datos: PostgreSQL
- Testing: JUnit para backend, Jasmine/Karma para frontend

Mecanismos del framework para patrones

- Inyección de dependencias (Spring): permite desacoplar la creación de instancias de los objetos, ideal para aplicar Strategy y Factory.
- Anotaciones (`@Component`, `@Service`, `@Autowired`): facilitan el uso de patrones y la gestión de beans.
- Spring Profiles o @Qualifier: útil para cambiar estrategias en Strategy dinámicamente.

Arquitectura de la Solución

La solución se basa en una arquitectura **MVC (Modelo-Vista-Controlador)** utilizando **Spring Boot** como framework principal. Esta arquitectura permite separar claramente las responsabilidades de cada capa del sistema, facilitando el mantenimiento, la escalabilidad y la prueba unitaria de los componentes.

- **Modelo**: Representa las entidades del dominio, como `PlanSuscripcion`, `EstrategiaPrecio`, `PlanBasicoFactory`, y encapsula la lógica de negocio. Aquí se aplican los patrones **Factory Method**, **Builder**, **Prototype** y **Strategy**, representando las diferentes formas de construir, clonar y calcular planes.

- **Vista**: Es desarrollada en Angular, encargada de la interacción con el usuario. Incluye pantallas para configurar el plan, ver un resumen en tiempo real y confirmar la suscripción. Estas vistas se comunican con el backend a través de servicios REST (`/crear-plan`, `/resumen`, `/confirmar`).

- **Controlador**: Se encarga de recibir las solicitudes HTTP desde el frontend y delegar la lógica al servicio correspondiente. Aquí se integran módulos como el gestor de suscripciones que usa los patrones de diseño para componer dinámicamente el plan del usuario.

Este enfoque MVC permite una implementación clara y desacoplada, en la que los controladores orquestan la lógica, los servicios encapsulan la aplicación de los patrones de diseño y las vistas permiten una experiencia rica al usuario. Además, el uso de **inyección de dependencias de Spring**, anotaciones como `@Component`, `@Service` y `@Autowired`, y la posibilidad de utilizar `@Qualifier` para cambiar estrategias en tiempo de ejecución, hacen que el sistema sea altamente mantenible y flexible.

Explicación detallada por patrón

- Factory Method: Permite crear instancias de planes de forma polimórfica. Las subclases de `PlanSuscripcionFactory` deciden qué tipo concreto retornar.

- Builder: Facilita la construcción de planes complejos con muchos atributos. Se encadena la configuración para mejorar la legibilidad y control.

- Prototype: A través de `clone()`, se reutilizan planes ya configurados. Útil para usuarios que desean repetir configuraciones con pequeñas variaciones.

- Strategy (nuevo): Se define una interfaz `EstrategiaPrecio` con distintas implementaciones (por ejemplo, `PrecioBase`, `PromocionEstudiante`, `DescuentoPorDispositivo`).
- La clase `PlanSuscripcion` puede tener un método `setEstrategia(EstrategiaPrecio estrategia)` y delegar el cálculo de precio a la estrategia activa.

Conclusiones

La implementación del sistema permitió poner en práctica diversos patrones de diseño creacionales como Factory Method, Builder y Prototype, así como el patrón de comportamiento Strategy. Gracias a una arquitectura modular (inspirada en la arquitectura hexagonal), se logró separar la lógica de negocio del acceso a datos y la interfaz, mejorando la mantenibilidad y escalabilidad del sistema. La integración del patrón Strategy aportó una solución flexible para gestionar diferentes formas de calcular precios dinámicos sin modificar la lógica del plan. El sistema está preparado para evolucionar fácilmente con nuevos planes, reglas y condiciones comerciales.

Lecciones Aprendidas

1. Comprender la importancia de los patrones de diseño en la construcción de sistemas extensibles y mantenibles.

2. El patrón Strategy permite cambiar el comportamiento sin modificar el núcleo del sistema.

3. La arquitectura hexagonal ayuda a desacoplar responsabilidades y facilita las pruebas unitarias.

4. El patrón Builder facilita la creación de objetos complejos y mejora la legibilidad del código.

5. Factory Method simplifica la creación de objetos según condiciones definidas sin acoplar el cliente a una clase específica.

6. Prototype resulta útil cuando se necesita duplicar objetos configurados para acelerar procesos similares.

7. Integrar patrones en un proyecto real demanda comprender cómo interactúan entre sí dentro de una arquitectura general.