Proyecto - Entrega 2

Diseño y Arquitectura de Algoritmos 2024-2

Aplicación de estilo y patrones arquitectónicos

1. Introducción

En esta segunda fase del proyecto, se han realizado modificaciones significativas en la arquitectura del sistema, pasando de una Arquitectura Orientada a Servicios (SOA) a un **Monolito en capas**. Esta decisión se tomó para optimizar el flujo de datos y reducir la complejidad en la gestión de servicios distribuidos, asegurando una implementación más simple y eficiente en esta etapa del desarrollo.

En paralelo al cambio arquitectónico, se ha modificado el **diseño del sistema** para implementar **patrones de diseño** que optimicen la relación entre clases y mejoren la flexibilidad del código. Estos patrones ayudan a gestionar la complejidad del sistema monolítico, aportando modularidad dentro del contexto de un solo bloque de código. La integración de patrones como ***Observer***, ***Singleton***, y ***Adapter*** garantiza que el sistema sea fácilmente mantenible y escalable.

Este documento detalla la estructura de la nueva arquitectura monolítica, el diseño de clases, los patrones de diseño implementados, y cómo cada uno contribuye a un flujo de datos más eficiente y coherente. Estos cambios nos preparan para evolucionar el sistema de manera flexible y responder a las necesidades futuras, manteniendo la estabilidad del desarrollo actual.

2. Estilo Arquitectónico

Durante la primera fase del proyecto, se planteó la implementación de una Arquitectura Orientada a Servicios (SOA), debido a la flexibilidad y escalabilidad que proporciona al dividir la aplicación en servicios independientes que pueden ser administrados y desplegados de manera aislada. Sin embargo, para esta segunda fase, se decidió adoptar una arquitectura monolítica, con el fin de simplificar el desarrollo, centralizar el control del sistema y reducir la complejidad que implica coordinar múltiples servicios distribuidos.

Arquitectura Monolítica

La arquitectura monolítica es un estilo arquitectónico en el que todas las funcionalidades de la aplicación están integradas y desplegadas como una única unidad. En este tipo de sistema, la lógica de negocio, la interfaz de usuario y el acceso a los datos están profundamente entrelazados y gestionados de manera centralizada. Esto facilita la comunicación interna entre componentes y simplifica la gestión de dependencias, ya que todos los módulos residen en el mismo espacio de memoria.

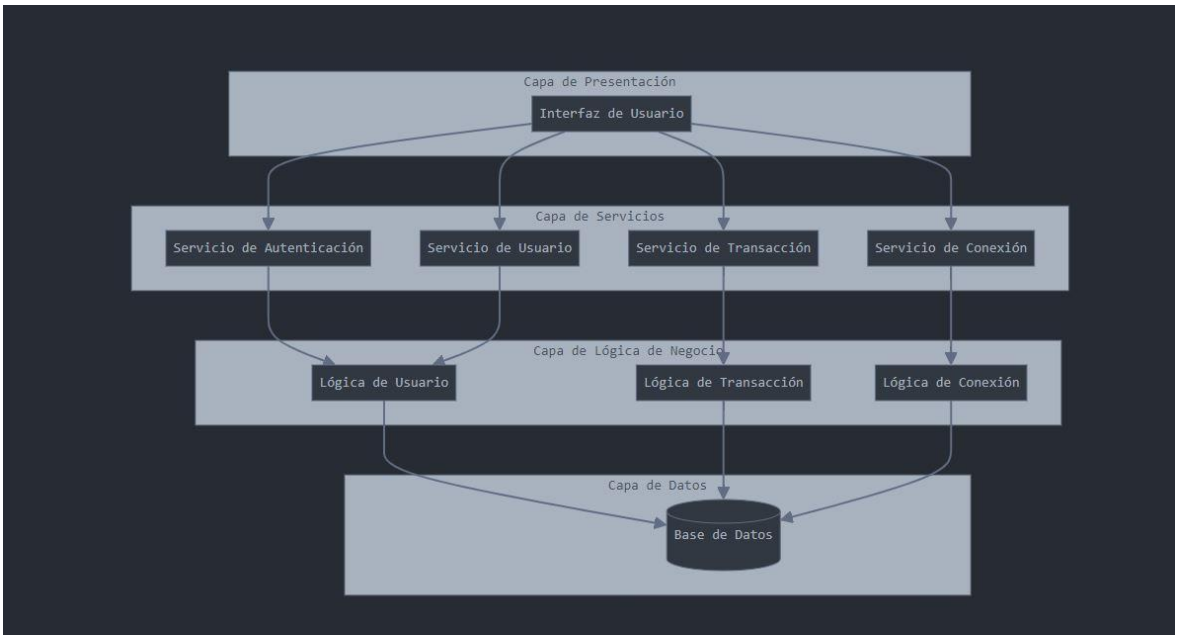
Carlos Bello 0000272648
Andrey Esteban Conejo 0000281295
Juan Miguel Dimaté 0000282752

Una de las principales ventajas de la arquitectura monolítica es que permite un desarrollo más rápido en las primeras etapas del proyecto, al evitar la sobrecarga de orquestar múltiples servicios y gestionar la comunicación entre ellos. Además, en términos de despliegue, solo es necesario lanzar una única aplicación, lo que reduce la complejidad en la infraestructura y el monitoreo.

El diseño de nuestro estilo arquitectónico está organizado por capas, lo que implica una separación clara de responsabilidades dentro del sistema. En una arquitectura monolítica por capas, la aplicación se divide en distintos niveles o capas, cada una con funciones específicas y bien delimitadas. Esta estructuración permite que cada capa se enfoque en su responsabilidad, facilitando el mantenimiento y la evolución del sistema, ya que los cambios en una capa no afectan directamente a las otras. Una de las principales ventajas de organizar el monolito por capas es que se logra un mayor orden y modularidad dentro de la aplicación, lo que contribuye a un código más limpio y fácil de mantener.

3. Diagramas:

Diagrama de componentes:



Explicación

Las líneas que conectan las clases indican las relaciones y dependencias entre ellas. Por ejemplo, la Interfaz de Usuario se conecta con los servicios en la capa de lógica de negocio, y estos servicios a su vez se conectan con la Base de Datos para realizar operaciones de almacenamiento y recuperación de datos, estas estan divididas entre 3 capas principales:

Carlos Bello 0000272648
Andrey Esteban Conejo 0000281295
Juan Miguel Dimaté 0000282752

- **Capa de Presentación**:

  Interfaz de Usuario: Esta clase representa la interfaz con la que interactúan los usuarios. Es responsable de mostrar la información y recibir las entradas del usuario.

- **Capa de Lógica de Negocio**:

  Servicio de Autenticación: Maneja la autenticación de usuarios.

  Servicio de Usuario: Gestiona la información y operaciones relacionadas con los usuarios.

  Servicio de Transacción: Se encarga de las operaciones de transacción.

  Servicio de Conexión: Administra las conexiones necesarias para el funcionamiento del sistema.

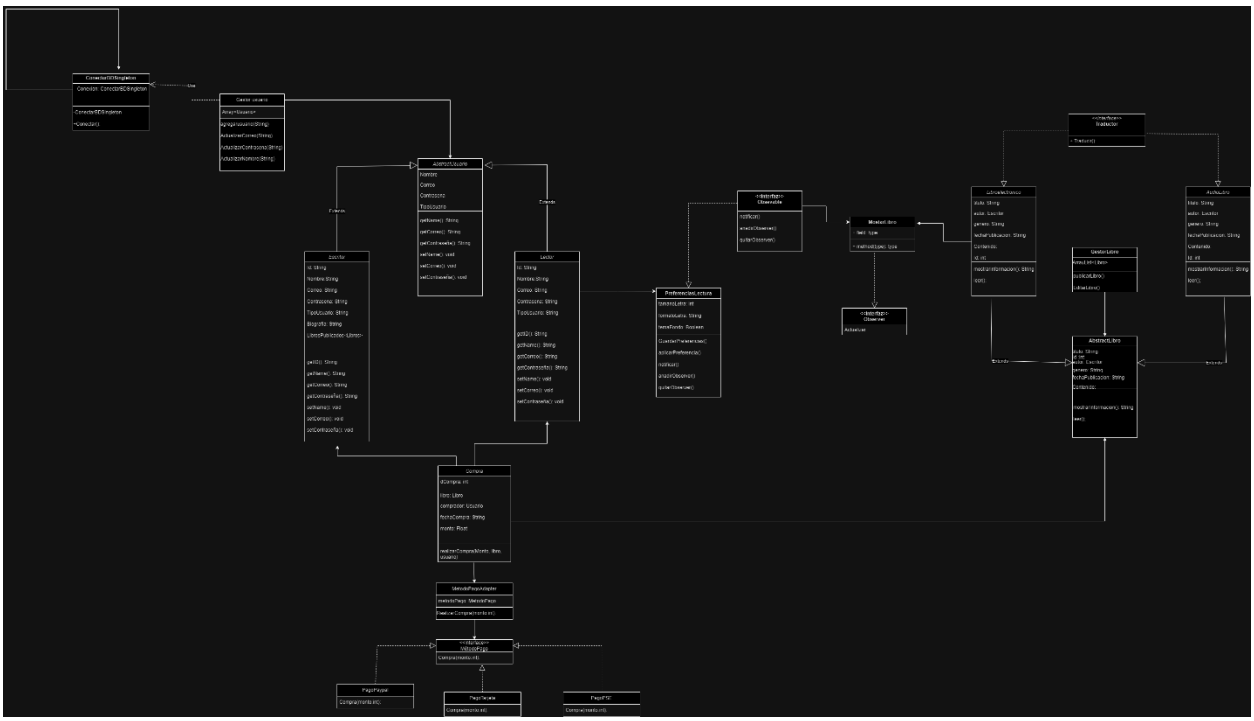
  Lógica de Usuario, Lógica de Transacción, Lógica de Conexión: Estas clases contienen la lógica específica para cada servicio, implementando las reglas de negocio y procesos necesarios.

- **Capa de Datos**:

  Base de Datos: Esta clase representa el almacenamiento de datos del sistema. Es donde se guardan y recuperan los datos necesarios para las operaciones del sistema.

Carlos Bello 0000272648
Andrey Esteban Conejo 0000281295
Juan Miguel Dimaté 0000282752

Diagrama de clases:



Conexiones entre Clases

- Usuario y Lector: Ambas clases tienen atributos similares y métodos para gestionar la información del usuario.
- Compra: Esta clase está conectada con Usuario, LibroElectronico, y MetodoPago. Utiliza estos para realizar la compra.
- MetodoPago: Es una interfaz que es implementada por PagoPaypal, PagoTarjeta, y PagoPSE. Esto permite que Compra pueda utilizar cualquier método de pago sin depender de una implementación específica.
- PreferenciasLectura: Implementa la interfaz Observable y puede notificar a los observadores (implementaciones de Observer) sobre cambios en las preferencias de lectura.
- LibroElectronico y AudioLibro: Ambas clases heredan de AbstractLibro y comparten atributos y métodos comunes.

Carlos Bello 0000272648
Andrey Esteban Conejo 0000281295
Juan Miguel Dimaté 0000282752

3. Implementación de patrones de diseño

3.1 Patrón Singleton

El patrón Singleton asegura que una clase tenga una única instancia a lo largo del ciclo de vida de la aplicación y proporciona un punto global de acceso a esa instancia. En nuestro proyecto, este patrón se utilizó para gestionar la conexión a la base de datos, dado que todas las interacciones con la base de datos deben realizarse a través de una única instancia compartida. Esto permite evitar la creación de múltiples conexiones innecesarias que podrían degradar el rendimiento o causar inconsistencias en los datos. En el diagrama, la clase ConexionDB es un claro ejemplo del uso de este patrón. Se garantiza que la conexión a la base de datos es única en toda la aplicación, garantizando la coherencia en las operaciones de lectura y escritura de datos.

3.2 Patrón Observer

El patrón Observer permite que un objeto (el observable) notifique a otros objetos (observadores) cuando hay cambios en su estado. En este proyecto, el patrón se utiliza en la clase PreferenciasUsuario, que maneja cómo los usuarios prefieren visualizar su contenido, por ejemplo, el tamaño de la fuente y el modo oscuro o claro.

Cuando un usuario cambia alguna preferencia, como el modo de visualización, la clase PreferenciasUsuario notifica a la clase MostrarLibro, encargada de adaptar la visualización de los libros electrónicos según las preferencias del usuario.

3.3 Patrón Adapter

El patrón Adapter permite que clases con interfaces incompatibles trabajen juntas. En este proyecto, la clase AdaptadorPago implementa el patrón Adapter para convertir las diferentes interfaces de los métodos de pago (tarjeta de crédito, PayPal, etc.) a una interfaz común que pueda ser utilizada por el sistema de transacciones del proyecto. permitiendo que el sistema procese diversas formas de pago sin necesidad de cambiar la lógica interna.

Carlos Bello 0000272648
Andrey Esteban Conejo 0000281295
Juan Miguel Dimaté 0000282752