UNIVERSIDAD LA SALLE

CARRERA DE INGENIERÍA DE SISTEMAS

ARQUITECTURA DE DESARROLLO EN CAPAS

Equipo de desarrollo: Ricardo Gabriel Barrientos García
Limber Huchani Aranda
Gustavo Amir Reyes Gonzales
Rosalía Reynaga Funes

Investigación realizada para
La materia Leguaje Audiovisual Sobre:
“Arquitectura de desarrollo en Capas”

La Paz - Bolivia
2023

1.Introducción

1.1. Contexto y Justificación

La Arquitectura de Desarrollo en Capas constituye un pilar esencial en la concepción y construcción de sistemas de software. Este enfoque arquitectónico ha demostrado ser crucial para la gestión eficiente y escalable de sistemas complejos en este campo de estudio. La asignación de esta investigación específica surge con el propósito de no solo comprender los fundamentos de esta arquitectura, sino también de explorar su aplicación y relevancia en el contexto del Lenguaje Audiovisual.

1.2. Objetivos de la Investigación

El objetivo principal de esta investigación es proporcionar un análisis exhaustivo de la Arquitectura de Desarrollo en Capas. Este estudio tiene como finalidad expandir nuestro conocimiento en la materialización de sistemas audiovisuales más eficaces y adaptables. A través de casos de estudio concretos, se busca ilustrar de qué manera esta arquitectura puede ser un elemento clave en la gestión de proyectos en este ámbito.

1.3. Alcance y Limitaciones

Esta investigación se centra en los principios, componentes y buenas prácticas asociadas con la Arquitectura de Desarrollo en Capas. Aunque se presentarán casos de estudio y ejemplos pertinentes, no se profundizará en implementaciones de lenguajes o tecnologías específicas. Además, se reconocen las limitaciones inherentes al enfoque en capas y se proporcionarán estrategias para mitigar los posibles desafíos durante la implementación.

2.Definición y Conceptos Básicos

2.1. ¿Qué es la Arquitectura de Desarrollo en Capas?

La Arquitectura de Desarrollo en Capas es un enfoque de diseño de software que organiza una aplicación en diferentes niveles o capas, cada una con una responsabilidad específica. La capa de presentación se encarga de la interfaz de usuario, la capa de lógica de negocio gestiona la funcionalidad y la capa de acceso a datos se encarga de interactuar con la base de datos o fuentes de información. Esta separación facilita el desarrollo, mantenimiento y escalabilidad del sistema.

2.2. Principios Fundamentales

Uno de los principios clave de esta arquitectura es la "Separación de Responsabilidades", que asegura que cada capa se concentre en una tarea particular, evitando la mezcla de lógica de presentación, negocio y acceso a datos. Esto promueve un código más limpio y mantenible, facilita la colaboración entre equipos de desarrollo además permite cambios en una capa sin afectar las demás. Otro principio es la "Reutilización de Componentes", lo que significa que los módulos o componentes desarrollados en una capa pueden ser aprovechados en otras partes del sistema, promoviendo la eficiencia y consistencia.

3.Historia y Evolución

3.1. Orígenes y Antecedentes

La Arquitectura de Desarrollo en Capas tiene sus raíces en la necesidad de gestionar la complejidad creciente de los sistemas de software. Se puede rastrear su origen en las primeras metodologías de diseño y programación estructurada de la década de 1970. En esta época, se buscaba organizar el código en módulos separados para facilitar la comprensión y el mantenimiento.

3.2. Desarrollo y Adopción en la Industria

A medida que la tecnología y las aplicaciones informáticas evolucionaron, la Arquitectura en Capas se convirtió en un enfoque central para el diseño de sistemas complejos. A finales del siglo

XX y principios del siglo XXI, empresas líderes en tecnología adoptaron este modelo para desarrollar aplicaciones de gran envergadura. Hoy en día, es un estándar en la industria del software y ha sido adaptado y refinado en diversas formas.

4.Ventajas y Beneficios

4.1. Razones para Adoptar una Arquitectura en Capas

La adopción de la Arquitectura de Desarrollo en Capas se justifica por varias razones. Permite una clara separación de responsabilidades, lo que facilita el desarrollo simultáneo de diferentes partes del sistema. Además, al promover la reutilización de componentes, se acelera el proceso de desarrollo y se reduce la redundancia de código. Esto se traduce en aplicaciones más eficientes y fáciles de mantener.

4.2. Mejoras en Mantenibilidad y Escalabilidad

Una de las ventajas más notables de esta arquitectura radica en su impacto en la mantenibilidad del sistema. Los cambios o actualizaciones en una capa pueden realizarse de forma aislada, minimizando el riesgo de efectos colaterales. Asimismo, la escalabilidad se ve beneficiada, ya que cada capa puede ser escalada de manera independiente, lo que facilita la gestión de cargas crecientes de usuarios o datos.

4.3. Facilitación de la Colaboración

La Arquitectura en Capas fomenta la colaboración efectiva entre equipos de desarrollo. Al dividir la aplicación en componentes claros y delimitados, diferentes equipos pueden trabajar en paralelo en áreas específicas sin interferir en las demás. Esto agiliza el proceso de desarrollo y permite la entrega de productos de alta calidad en tiempos más cortos.

5.Componentes de la Arquitectura en Capas

5.1. Capa de Acceso a Datos

La "Capa de Acceso a Datos" es una parte crucial en la arquitectura de capas en ingeniería de software, también conocida como “Capa de Acceso a la Base de Datos". Esta capa se centra en la gestión y manipulación de los datos que son almacenados y recuperados desde una fuente de datos, como una base de datos y archivos y dando una breve explicación de lo que se trata podríamos decir:

5.1.1. Propósito principal:

La Capa de Acceso a Datos tiene como objetivo principal abstraer la lógica de almacenamiento y recuperación de datos, permitiendo que las capas superiores del sistema (como la lógica de negocio y la interfaz de usuario) accedan a los datos de manera eficiente y sin necesidad de conocer los detalles de cómo se almacenan o recuperan.

5.1.2. Funciones

- Conexión y comunicación con la fuente de datos: Maneja la conexión a la fuente de datos, ya sea una base de datos, un archivo, un servicio web, etc.
- Operaciones de lectura y escritura: Proporciona métodos para consultar e insertar, actualizar, eliminar datos en la fuente de datos.
- Mapeo de objeto-relacional (ORM): En aplicaciones que utilizan bases de datos relacionales, esta capa puede incluir el mapeo entre objetos en el código y las tablas en la base de datos.
- Consultas: Puede incluir lógica para optimizar consultas y transacciones con el fin de mejorar el rendimiento y la eficiencia del sistema.

5.1.3. Abstracción y desacoplamiento

La Capa de Acceso a Datos se diseña para proporcionar una interfaz abstraída que oculta la complejidad de las operaciones de acceso a datos. Esto permite que otras partes del sistema interactúen con los datos sin necesidad de conocer los detalles de implementación de la fuente de datos.

5.2. Capa de Negocio

La Capa de Negocio es la encargada de abstraer las reglas del negocio en la funcionalidad y algoritmos que componen la lógica de la aplicación. Encargada de procesamiento y transformación de la información dada por el usuario o la almacenada en el sistema, esta capa se ocupa de mantener independiente la lógica mientras proporciona todos los servicios de manipulación de datos.

5.2.1. Propósito principal

El principal propósito de esta capa es proporcionar y aplicar la mayor parte de la lógica de la aplicación. La Capa de Negocio se ocupa de cumplir con las reglas del negocio convirtiéndolas en funciones y algoritmos que la aplicación usa para procesar los datos dentro de la misma. Además, esta capa existe para mantener la parte de la lógica del negocio independiente de las capas de Presentación y de Datos, de manera que la misma pueda reutilizarse en contextos varios y conectarse a distintos sistemas de almacenamiento de datos.

5.2.2. Funciones

- Definir las reglas bajo las que funciona la aplicación, abstrayendo la lógica del negocio a algoritmos con los cuales los datos puedan ser procesados y/o transformados.
- Permite a los desarrolladores diseñar aplicaciones compatibles con múltiples interfaces de usuario, lo que minimiza las posibilidades de duplicación innecesaria de código.

- Mantiene la lógica de la aplicación independiente de la tecnología e interfaces, esto con el objetivo de hacer de esa lógica adaptable a distintos entornos, pudiendo reutilizarla cuantas veces sea necesario.

5.3. Capa de Presentación

La Capa de Presentación constituye la interfaz a través de la cual los usuarios interactúan con la aplicación. Puede adoptar diversas formas, como interfaces gráficas en aplicaciones de escritorio o interfaces web en aplicaciones basadas en navegador. Es en esta capa donde se diseñan y presentan los elementos visuales, como botones, formularios y gráficos, para facilitar la interacción del usuario con la aplicación.

5.3.1. Propósito Principal

El propósito fundamental de la Capa de Presentación es proporcionar una interfaz intuitiva y amigable para que los usuarios interactúen con la aplicación de manera efectiva. Su diseño y disposición de elementos visuales influyen directamente en la experiencia del usuario, por lo que es crucial que sea atractiva y funcional.

5.3.2. Funciones

- La Capa de Presentación se encarga de la disposición y diseño de los elementos visuales, asegurando que la interfaz sea clara y fácil de usar.
- Captura y procesa la entrada del usuario, ya sea a través de clics de mouse, entradas de teclado o gestos táctiles, dependiendo del tipo de interfaz.
- Despliega la información de manera legible y coherente para el usuario, utilizando elementos como formularios, tablas y gráficos.
- Puede realizar validaciones en la entrada del usuario para asegurarse de que cumple con los requisitos del sistema.

5.4. Interconexión entre Capas

La Interconexión entre Capas define cómo las diferentes capas de la arquitectura se comunican entre sí. Esto incluye el intercambio de datos y la coordinación de operaciones entre la Capa de Presentación, la Capa de Lógica de Negocio y la Capa de Acceso a Datos.

5.4.1. Propósito Principal

El propósito de la Interconexión entre Capas es garantizar una comunicación eficiente y segura entre las diferentes partes del sistema. Esto permite la sincronización de operaciones y la transferencia de datos necesarios para el funcionamiento cohesivo de la aplicación.

5.4.2. Funciones

- Define los protocolos y métodos de intercambio de datos entre las capas, como HTTP en aplicaciones web o sockets en aplicaciones de red.
- Facilita el envío y recepción de datos entre la Capa de Presentación y la Capa de Lógica de Negocio, así como entre la Capa de Lógica de Negocio y la Capa de Acceso a Datos.
- Puede incluir mecanismos para garantizar la integridad de los datos durante su transmisión y asegurar la autenticación y autorización adecuadas.

5.5. Grafica de funcionamiento

El gráfico a continuación ilustrará de manera visual la arquitectura en capas y su funcionamiento.

Ver figura 1

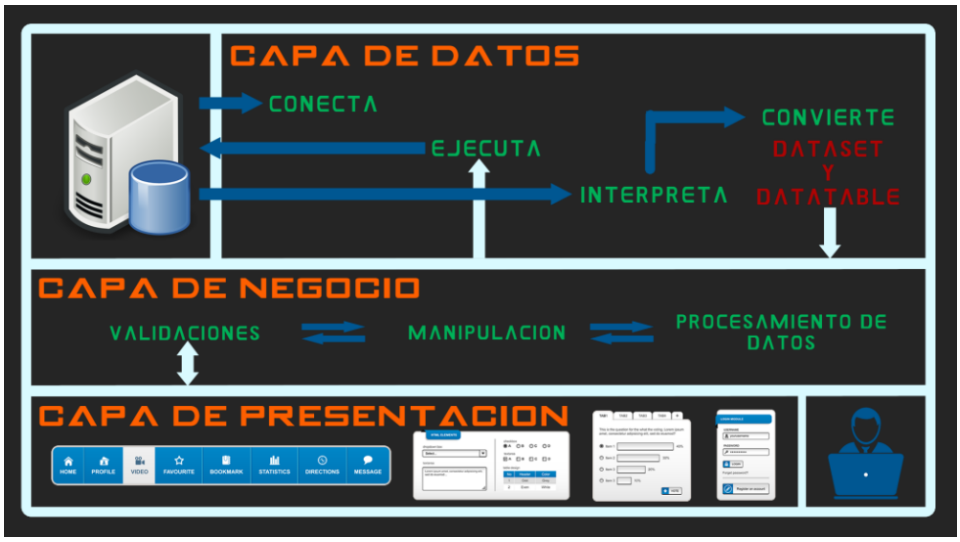

Figura 1 Funcionamiento

6.Patrones y Modelos Relacionados

- MVC (Modelo-Vista-Controlador): MVC es un patrón de diseño que separa la aplicación en tres componentes principales: Modelo (representación de los datos), Vista (interfaz de usuario) y Controlador (gestión de la interacción del usuario). Este patrón promueve una organización clara y modular del código. Ver figura 2

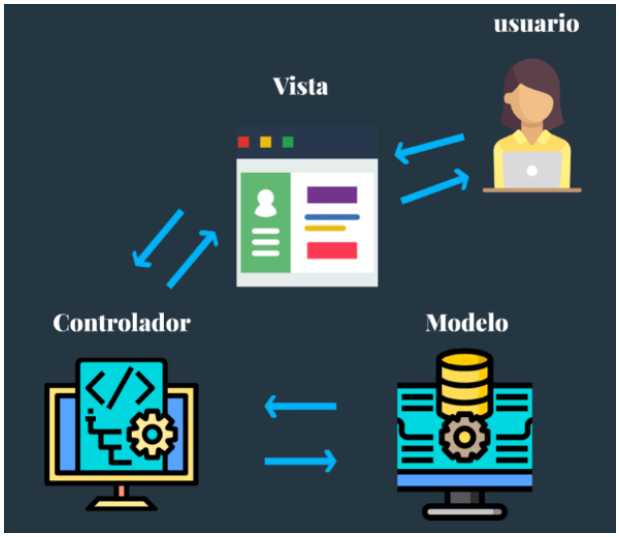

Figura 2 MVC

- MVVM (Modelo-Vista-VistaModelo): MVVM es un patrón que también separa la aplicación en tres componentes: Modelo (datos y lógica), Vista (interfaz de usuario) y VistaModelo (maneja la lógica de presentación). MVVM se enfoca en la vinculación de datos y facilita la actualización automática de la interfaz. Ver figura 3

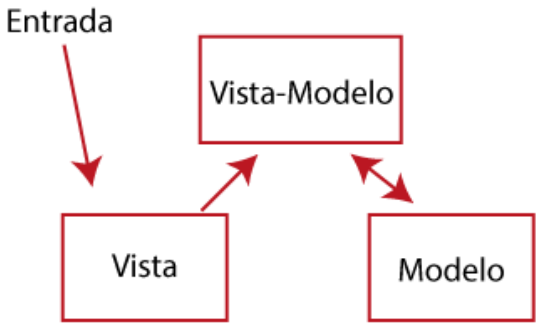

Figura 3 MVVM

- N-Tier (N-Capas): N-Tier es una arquitectura que divide una aplicación en capas lógicas o niveles, cada una con funciones específicas. Esto facilita la escalabilidad y mantenimiento de grandes aplicaciones, al mantener una clara separación de responsabilidades. Ver figura 4

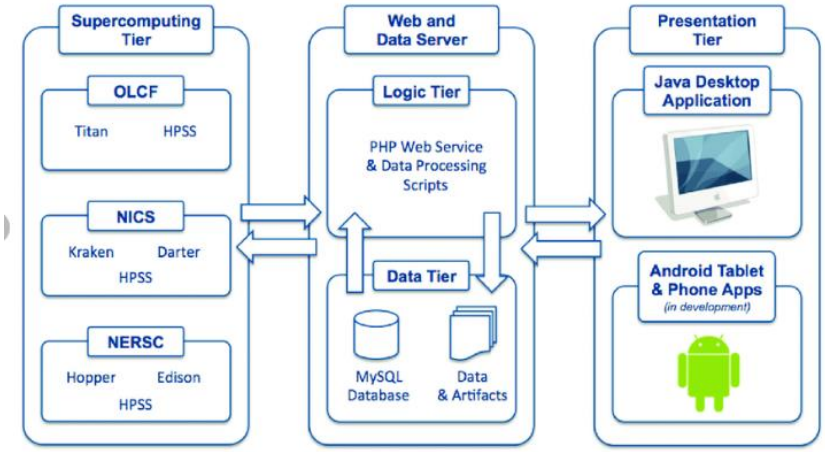

Figura 4 N-capas

7.Implementación Práctica

7.1. Ejemplos y Casos de Uso

La implementación práctica de la Arquitectura de Desarrollo en Capas se puede ilustrar con ejemplos concretos. Por ejemplo, en una aplicación de comercio electrónico, la Capa de Presentación se encargaría de la interfaz de usuario donde los usuarios navegan y realizan compras. La Capa de Lógica de Negocio gestionaría procesos como el carrito de compras, el cálculo de precios y la gestión de inventario. Finalmente, la Capa de Acceso a Datos se encargaría de interactuar con la base de datos para almacenar y recuperar información de productos y usuarios.

7.2. Herramientas y Tecnologías Comunes

La implementación de esta arquitectura se apoya en una amplia gama de herramientas y tecnologías. Frameworks como Spring para Java o ASP.NET para C# proporcionan estructuras que facilitan la creación y gestión de las distintas capas. Además, lenguajes como JavaScript, Python o Ruby, junto con bibliotecas como React o Angular, son fundamentales para el desarrollo de la Capa de Presentación en aplicaciones web.

7.3. Ejemplo Práctico

Para explicar esta arquitectura de forma práctica utilizaremos el caso de una página web. Lo primero que se debe hacer es identificar las responsabilidades de cada capa. La capa de presentación (Ver figura 5) se encargará de mostrar la información al usuario final, la capa de negocio (Ver figura 6) procesará la información y la capa de datos (Ver figura 7) se hará cargo de almacenar y recuperar la información

```html
<!DOCTYPE HTML>

<HTML>

<HEAD>

<TITLE>MY PAGE</TITLE>

<SCRIPT SRC="MYSCRIPT.JS"></SCRIPT>

</HEAD>

<BODY>

<H1>WELCOME TO MY PAGE</H1>

<FORM>

<LABEL FOR="NAME">NAME:</LABEL>

<INPUT TYPE="TEXT" ID="NAME" NAME="NAME"><BR>
<BR>

<LABEL FOR="EMAIL">EMAIL:</LABEL>

<INPUT TYPE="EMAIL" ID="EMAIL" NAME="EMAIL">
<BR><BR>
<BUTTON ONCLICK="SUBMITFORM()">SUBMIT</BUTTON>

</FORM>

</BODY>

</HTML>
```

Figura 5 Código HTML: Capa de presentación

```
PUBLIC CLASS USER {

    PRIVATE STRING NAME;

    PRIVATE STRING EMAIL;

    PUBLIC USER(STRING NAME, STRING EMAIL) {

    THIS.NAME = NAME;

    THIS.EMAIL = EMAIL;

    }

    PUBLIC STRING GETNAME() {

    RETURN NAME;

    }

    PUBLIC STRING GETEMAIL() {

    RETURN EMAIL;

    }

    }

    PUBLIC CLASS USERSERVICE {

    PUBLIC USER CREATEUSER(STRING NAME, STRING EMAIL) {

    // VALIDACIÓN Y PROCESAMIENTO DE DATOS AQUÍ

    RETURN NEW USER(NAME, EMAIL);

    }

    }
```

Figura 6 Código: Java Capa de negocio

```
PUBLIC CLASS USERDAO {

    PUBLIC VOID SAVEUSER(USER USER) {

// LÓGICA DE ALMACENAMIENTO EN BASE DE DATOS AQUÍ

    }

}
```

Figura 7 Código: Java Capa de acceso de datos

A continuación, debemos definir las interfaces por capas para asegurar la separación de responsabilidades. En este caso incluimos la interfaz de la capa de negocio (Ver figura 8)

```
PUBLIC CLASS USERSERVICE {

    PUBLIC USER CREATEUSER(STRING NAME, STRING EMAIL) {

    // VALIDACIÓN Y PROCESAMIENTO DE DATOS AQUÍ

    RETURN NEW USER(NAME, EMAIL);
    }
}
```

Figura 8 Interfaz Capa de negocio

Finalmente incluimos los servicios y los aplicamos, cada nivel debe ofrecer servicios particulares para llevar a cabo sus funciones específicas. Por ejemplo, en la capa de presentación, utilizamos el servicio USER.CREATEUSER de la capa de negocio para crear un usuario y el servicio USERDAO.SAVEUSER de la capa de acceso de datos (Ver figura 9). De esta forma usamos todas las capas en la capa de presentación. En la capa de negocio, se utilizan servicios para procesar información y realizar cálculos más avanzados (Ver figura 10). Mientras que, en la capa de acceso a datos, se implementan servicios para almacenar y recuperar datos desde la base de datos. (Ver figura 11)

```javascript
function submitForm() {
    var name = document.getElementById("NAME").value;
    var email = document.getElementById("EMAIL").value;

    // Obtención de valores del formulario
    // Los valores de los campos de entrada con IDs "NAME" y "EMAIL" se asignan a las variables name y email

    // LLAMADA A SERVICIO DE LA CAPA DE LÓGICA DE NEGOCIO
    // Se utiliza el servicio USERSERVICE.CREATEUSER para crear un usuario con los datos name y email
    var user = USERSERVICE.CREATEUSER(name, email);

    // LLAMADA A SERVICIO DE LA CAPA DE ACCESO A DATOS
    // Se utiliza el servicio USERDAO.SAVEUSER para guardar el usuario en la capa de acceso a datos
    USERDAO.SAVEUSER(user)
}

```

Figura 9 Servicio: Capa de presentación

```
PUBLIC CLASS USERSERVICE {

    PUBLIC STATIC USER CREATEUSER(STRING NAME, STRING EMAIL) {

// VALIDACIÓN Y PROCESAMIENTO DE DATOS AQUÍ

        RETURN NEW USER(NAME, EMAIL);

    }

}
```

Figura 10 Servicio: Capa de negocio

```
PUBLIC CLASS USERDAO {

    PUBLIC STATIC VOID SAVEUSER(USER USER) {

// LÓGICA DE ALMACENAMIENTO EN BASE DE DATOS AQUÍ

    }

}
```

Figura 11 Servicio: Capa de acceso de datos

8.Desafíos y Consideraciones

8.1. Posibles Problemas y Desventajas

A pesar de sus ventajas, la Arquitectura en Capas presenta desafíos potenciales. Una mala definición de las interfaces entre capas puede llevar a una comunicación ineficiente o a dependencias excesivas. Además, una excesiva subdivisión de capas puede conducir a una complejidad innecesaria y dificultar la comprensión del sistema en su conjunto. Es importante encontrar el equilibrio adecuado entre el modularidad y la simplicidad.

8.2. Estrategias para Mitigar Riesgos

Para abordar estos desafíos, es crucial implementar estrategias adecuadas. Esto puede incluir una cuidadosa planificación y diseño de las interfaces entre capas, así como la adopción de patrones de diseño que promuevan una comunicación clara y eficiente. Además, es fundamental realizar pruebas exhaustivas para identificar y corregir posibles problemas de integración entre las capas.

8.3. Casos Especiales y Soluciones Alternativas

En ciertos escenarios, la Arquitectura en Capas puede no ser la solución óptima. Por ejemplo, en aplicaciones simples o de tamaño reducido, una arquitectura menos compleja podría ser más adecuada. Además, en entornos de microservicios, donde la independencia y escalabilidad de los componentes son críticas, otras arquitecturas como la de microservicios pueden ser preferibles.

9.Comparación con Otras Arquitecturas

9.1. Diferencias con Arquitecturas Monolíticas

En comparación con las arquitecturas monolíticas, la Arquitectura en Capas ofrece una mayor modularidad y flexibilidad. Las arquitecturas monolíticas tienden a integrar todas las funcionalidades en una sola aplicación, lo que puede llevar a un código más complejo y difícil de

mantener. En contraste, la Arquitectura en Capas permite una separación más clara de responsabilidades y una gestión más efectiva de los componentes.

9.2. Ventajas frente a Arquitecturas de Microservicios, etc.

En comparación con arquitecturas como la de microservicios, la Arquitectura en Capas puede ser más adecuada para aplicaciones menos complejas o de menor escala. Las arquitecturas de microservicios están diseñadas para sistemas altamente distribuidos y escalables, lo que puede introducir una complejidad adicional. La Arquitectura en Capas ofrece un equilibrio entre el modularidad y la simplicidad, lo que la hace más apropiada para ciertos contextos.

10.Estudios de Caso y Ejemplos Reales

10.1. Aplicaciones o Proyectos que Utilizan esta Arquitectura

Un ejemplo destacado de la implementación de la Arquitectura en Capas es el sistema de gestión de contenido WordPress. En este caso, la Capa de Presentación se refiere a la interfaz de administrador y los temas visuales que los usuarios interactúan para crear y gestionar contenido. La Capa de Lógica de Negocio se ocupa de la administración de usuarios, la gestión de contenido y la funcionalidad de los plugin`s. La Capa de Acceso a Datos interactúa con la base de datos MySQL para almacenar y recuperar la información.

10.2. Lecciones Aprendidas y Mejores Prácticas

Un aspecto crucial es entender las lecciones aprendidas de aplicaciones exitosas que implementan la Arquitectura en Capas. La gestión efectiva de las dependencias entre capas y la definición clara de interfaces son fundamentales para el éxito. Además, la modularidad adecuado y la asignación de responsabilidades son prácticas esenciales. La planificación y el diseño cuidadosos desde el inicio del proyecto también son clave para una implementación exitosa.

11.Conclusiones y Recomendaciones

11.1. Resumen de los Hallazgos Principales

La adopción de la Arquitectura en Capas ha demostrado ser una estrategia efectiva en el desarrollo de software, especialmente en proyectos de cierta complejidad. La clara separación de responsabilidades entre capas facilita el desarrollo y mantenimiento, permitiendo a los equipos trabajar de manera más eficiente y escalable. Además, el modularidad inherente a esta arquitectura facilita la reutilización de componentes y la adaptación a futuros cambios.

11.2. Recomendaciones para la Implementación

Para una implementación exitosa de la Arquitectura en Capas, es crucial seguir algunas recomendaciones clave. En primer lugar, una planificación detallada y un diseño cuidadoso de las interfaces entre capas son fundamentales. Además, se debe mantener una comunicación efectiva entre los equipos que trabajan en cada capa para garantizar una integración sin problemas. También es importante realizar pruebas exhaustivas para identificar y corregir posibles problemas de integración.

12.Referencias y Bibliografía

12.1. Fuentes Consultadas y Citas Relevantes

12.1.1. Fuentes

- Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley.
- Ambler, S. W. (2004). Introduction to Agile Modeling. Agile Modeling.
- Larman, C. (2004). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design. Prentice Hall

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
- Martin, R. C. (2000). Design Principles and Design Patterns. C++ Report, 13(4), 38-43.
- Shalloway, A., & Trott, J. R. (2001). Design patterns explained: A new perspective on object-oriented design. Addison-Wesley.
- Canal videojuegosydesarrollos de youtube video “capa de datos”
- Durán, M. (2023, abril 11). Qué es la arquitectura en capas, ventajas y ejemplos. Recuperado el 7 de octubre de 2023, de Hubspot.es website: https://blog.hubspot.es/website/que-es-arquitectura-en-capas

12.1.2. Citas

- Fowler (2002) señala que el patrón de arquitectura de capas es esencial para la organización eficiente de aplicaciones empresariales.
- Según Larman (2004), la separación de responsabilidades en la arquitectura en capas es clave para la mantenibilidad y escalabilidad del sistema.
- Gamma et al. (1994) describen en detalle varios patrones de diseño que son aplicables en el contexto de la arquitectura en capas.
- Ambler (2004) argumenta que el modelado ágil es una herramienta valiosa para la visualización y diseño de arquitecturas en capas.
- Martin (2000) destaca la importancia de los principios de diseño en la implementación efectiva de la arquitectura en capas.
- Shalloway y Trott (2001) proporcionan una perspectiva comprensible sobre la aplicación de patrones de diseño en la arquitectura en capas.