**Universidad La Salle – Bolivia**

**Carrera de Ingeniería de Sistemas**



**Tema:**

Arquitectura de Desarrollo de Software DDD - Domain-Driven Design

**Estudiantes:**

* Rene Manuel Avilés.
* José Milton Mamani.
* Roberto Carlos Aruquipa Arzala.
* Diego Carlos Rojas Apaza.

**Materia:**

ISI- 356 Lenguaje Audiovisual

**Docente:**  
Luis Jiménez Peña

**Gestión:**

II/23

**ÍNDICE**

[Arquitectura de Desarrollo de Software DDD - Domain-Driven Design 3](#_Toc147535314)

[Introducción 3](#_Toc147535315)

[Objetivos: 3](#_Toc147535316)

[Desafíos y Consideraciones: 3](#_Toc147535317)

[Marco Teórico: 4](#_Toc147535318)

[Proceso de Diseño con DDD: 4](#_Toc147535319)

[Implementación de DDD: 5](#_Toc147535320)

[Patrones y Buenas Prácticas de DDD: 5](#_Toc147535321)

[Integración con Otros Aspectos del Desarrollo: 6](#_Toc147535322)

[Herramientas y tecnologías: 6](#_Toc147535323)

[Marco Practico: 7](#_Toc147535324)

[Clase Reservation 7](#_Toc147535325)

[Clase ReservatiopsService 7](#_Toc147535326)

[Clase ReservationController 8](#_Toc147535327)

[Conclusiones: 10](#_Toc147535328)

[Bibliografía: 10](#_Toc147535329)

Índice De Ilustraciones

[Ilustración 1 Código Clase Reservation 8](#_Toc147535358)

[Ilustración 2 Código Clase ReservatiopsService 9](#_Toc147535359)

[Ilustración 3 Codigo Clase ReservationController 10](#_Toc147535360)

# Arquitectura de Desarrollo de Software DDD - Domain-Driven Design

## Introducción

Domain-Driven Design (DDD) es un enfoque de desarrollo de software que enfatiza en la importancia del modelo de dominio en el diseño y la implementación de sistemas de software. DDD se basa en la idea de que el modelo de dominio debe ser el foco central del proceso de desarrollo, y que el modelo debe basarse en una comprensión profunda del dominio del problema. Este enfoque ha sido reconocido como un tema crítico durante al menos 20 años, sin embargo, sorprendentemente, poco se ha escrito sobre lo que hay que hacer o cómo hacerlo.

## Objetivos:

* Comprender el dominio: Comprender en profundidad el dominio del problema, incluidas las normas empresariales, los procesos y la terminología.
* Crear un modelo de dominio: Crear un modelo conceptual del dominio del problema, utilizando un lenguaje común que compartan los desarrolladores y los expertos del dominio.
* Perfeccionamiento del modelo de dominio: Perfeccionar el modelo de dominio a través de un proceso iterativo de retroalimentación y refinamiento, para garantizar que refleja con precisión el dominio del problema.
* Implementación del modelo de dominio: Implementar el modelo de dominio en código, utilizando patrones y buenas prácticas para garantizar que el código está bien diseñado y es fácil de mantener.

## Desafíos y Consideraciones:

El implementar la arquitectura de desarrollo de software DDD (Domain-Driven Design)

* Complejidad del Dominio: DDD es más eficaz cuando se aplica a dominios complejos, pero esto también puede hacer que sea más difícil de implementar.
* Expertos en la materia: Trabajar con expertos en el dominio puede ser un reto, ya que pueden no tener una formación técnica y pueden tener dificultades para expresar sus ideas de una manera que sea comprensible para los desarrolladores.
* Colaboración del Equipo: DDD requiere una estrecha colaboración entre los desarrolladores y los expertos del dominio, lo que puede ser difícil de lograr en la práctica.
* Comunicación: La comunicación efectiva entre los equipos técnicos y los expertos en el dominio es crucial, pero puede ser desafiante.
* Cambios Constantes: DDD requiere que los desarrolladores aprendan nuevos patrones y prácticas, lo que puede llevar tiempo y esfuerzo.

## Marco Teórico:

Domain-Driven Design (DDD) es una aproximación holística al diseño de software que pone en el centro el dominio del negocio como faro del proyecto y en su modelo como herramienta de comunicación entre negocio y tecnología. DDD fue propuesta por primera vez por Eric Evans, quien es un líder de pensamiento en diseño de software y autor del libro “Domain-Driven Design”.

El diseño basado en dominios (DDD) es un enfoque para el desarrollo de software que enfatiza la comprensión del dominio o entorno comercial de un sistema de software y traduce esa comprensión en un modelo que se puede usar para guiar el diseño y desarrollo del sistema. Los principios de DDD guían a los arquitectos de software al proporcionar un conjunto de pautas y prácticas para desarrollar software que satisfaga las necesidades de la empresa y sus usuarios.

### Proceso de Diseño con DDD:

El proceso de diseño de software con DDD implica varios pasos, entre los que se incluyen:

* Comprensión del dominio: Obtener un conocimiento profundo del dominio del problema, incluidas las reglas de negocio, los procesos y la terminología.
* Identificar el núcleo del dominio: Identificar las partes más importantes y complejas del dominio del problema, y centrarse en estas áreas en el proceso de diseño.
* Creación de un modelo de dominio: Crear un modelo conceptual del dominio del problema, utilizando un lenguaje común que compartan los desarrolladores y los expertos del dominio.
* Perfeccionamiento del modelo de dominio: Perfeccionar el modelo de dominio a través de un proceso iterativo de retroalimentación y refinamiento, para garantizar que refleja con precisión el dominio del problema.
* Implementación del modelo de dominio: Implementar el modelo de dominio en código, utilizando patrones y buenas prácticas para garantizar que el código está bien diseñado y es fácil de mantener.

### Implementación de DDD:

La implementación de DDD implica el uso de patrones y mejores prácticas para garantizar que el modelo de dominio esté bien diseñado y bien implementado. Algunas estrategias clave de implementación son

* Uso de agregados: Los agregados son agrupaciones de objetos relacionados que se tratan como una única unidad de datos. El uso de agregados puede ayudar a simplificar el modelo de dominio y facilitar su comprensión y mantenimiento.
* Utilización de repositorios: Los repositorios son objetos que permiten almacenar y recuperar objetos de dominio. El uso de repositorios puede ayudar a desacoplar el modelo de dominio de la capa de persistencia, facilitando el cambio del mecanismo de persistencia en caso necesario.
* Uso de eventos de dominio: Los eventos de dominio son mensajes que se envían cuando ocurre algo importante en el modelo de dominio. El uso de eventos de dominio puede ayudar a desacoplar diferentes partes del modelo de dominio, facilitando el mantenimiento y la evolución del sistema a lo largo del tiempo.
* Uso de contextos delimitados: Los contextos delimitados son áreas autocontenidas del modelo de dominio que tienen un límite claro y un contexto bien definido. El uso de contextos delimitados puede ayudar a simplificar el modelo de dominio y facilitar su comprensión y mantenimiento.

### Patrones y Buenas Prácticas de DDD:

DDD hace hincapié en el uso de patrones y mejores prácticas para garantizar que el modelo de dominio está bien diseñado y bien implementado. Algunos patrones y prácticas clave son

* Lenguaje ubicuo: Esto implica el uso de un lenguaje común que comparten los desarrolladores y los expertos en el dominio, para garantizar que todos tengan una comprensión clara del dominio del problema.
* Agregados: Los agregados son agrupaciones de objetos relacionados que se tratan como una única unidad de datos. El uso de agregados puede ayudar a simplificar el modelo de dominio y facilitar su comprensión y mantenimiento.
* Repositorios: Los repositorios son objetos que permiten almacenar y recuperar objetos de dominio. El uso de repositorios puede ayudar a desacoplar el modelo de dominio de la capa de persistencia, facilitando el cambio del mecanismo de persistencia en caso necesario.
* Eventos de dominio: Los eventos de dominio son mensajes que se envían cuando ocurre algo importante en el modelo de dominio. El uso de eventos de dominio puede ayudar a desacoplar diferentes partes del modelo de dominio, haciéndolo más fácil.
* Entity: Representa un objeto con una identidad única, que permite el seguimiento a través de diferentes estados.

### Integración con Otros Aspectos del Desarrollo:

DDD puede integrarse con otros aspectos del desarrollo de software, como las metodologías de desarrollo ágil, el desarrollo basado en pruebas y la integración continua. DDD también se puede utilizar junto con otros patrones arquitectónicos, como los microservicios y la arquitectura basada en eventos. La clave es asegurarse de que el modelo de dominio sigue siendo el foco central del proceso de desarrollo, y que otros aspectos del desarrollo están alineados con el modelo de dominio. Algunos ellos pueden ser:

* Arquitectura Hexagonal: Separación clara entre la lógica del dominio y los detalles de implementación, permitiendo una fácil integración con diferentes capas y tecnologías.
* Microservicios: Implementación de DDD en un entorno de microservicios, donde cada microservicio representa un dominio del negocio independiente.
* Integración Continua y Despliegue Continuo: Prácticas que permiten una rápida retroalimentación y adaptación del modelo de dominio

### Herramientas y tecnologías:

Existen varias herramientas y tecnologías que pueden utilizarse para apoyar el DDD, como:

* Lenguajes específicos de dominio: Los lenguajes específicos de dominio (DSL) son lenguajes de programación diseñados para expresar conceptos y reglas de un dominio de problemas específico. El uso de un DSL puede ayudar a garantizar que el código refleje con precisión el dominio del problema.
* Marcos de mapeo objeto-relacional: Los marcos de mapeo objeto-relacional (ORM) proporcionan una forma de mapear objetos de dominio a tablas de bases de datos relacionales. El uso de un ORM puede ayudar a simplificar la capa de persistencia y facilitar el cambio del mecanismo de persistencia en caso necesario.
* Marcos de abastecimiento de eventos: Los marcos de abastecimiento de eventos proporcionan una forma de almacenar y recuperar eventos de dominio, que pueden utilizarse para reconstruir el estado del modelo de dominio en cualquier momento. El uso de fuentes de eventos puede ayudar a garantizar que el modelo de dominio refleje con precisión la historia del sistema.
* Herramientas de contenedorización y orquestación: Las herramientas de contenedorización y orquestación, como Docker y Kubernetes, se pueden utilizar para desplegar y gestionar arquitecturas basadas en microservicios que estén alineadas con el modelo de dominio.

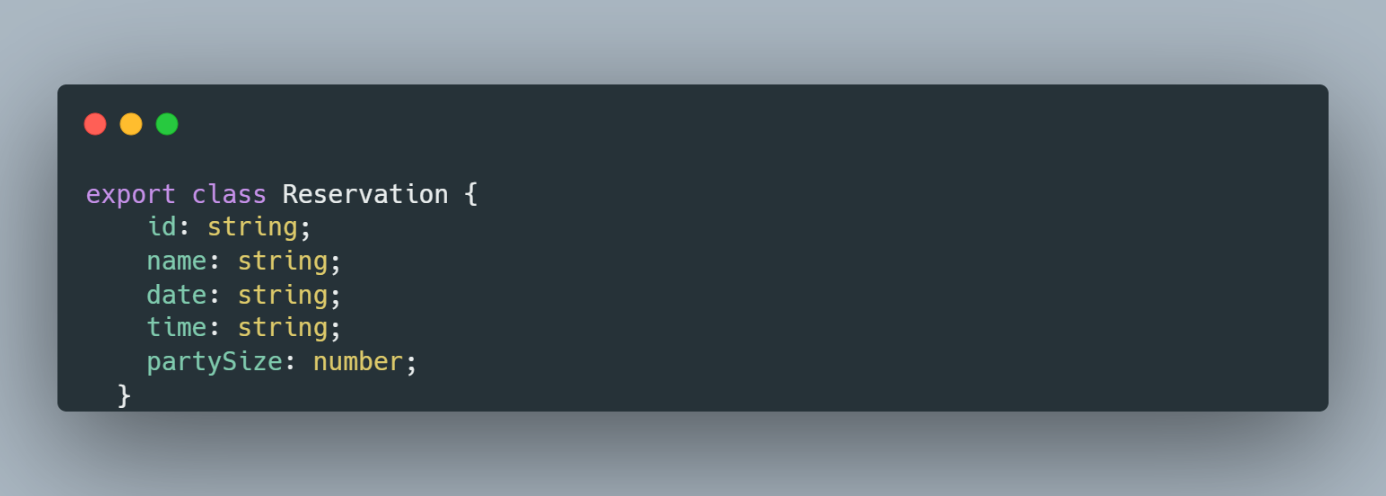
## Marco Practico:

A continuación, entraremos en la fase práctica en la que demostraremos la aplicación de Domain-Driven Design (DDD) mediante código. Este enfoque práctico nos permitirá ilustrar cómo los principios y patrones de DDD se traducen en la implementación de un sistema de software.

### Clase Reservation

Pequeña explicacion

Ilustración Código Clase Reservation



Fuente: Elaboración propia

### Clase ReservatiopsService

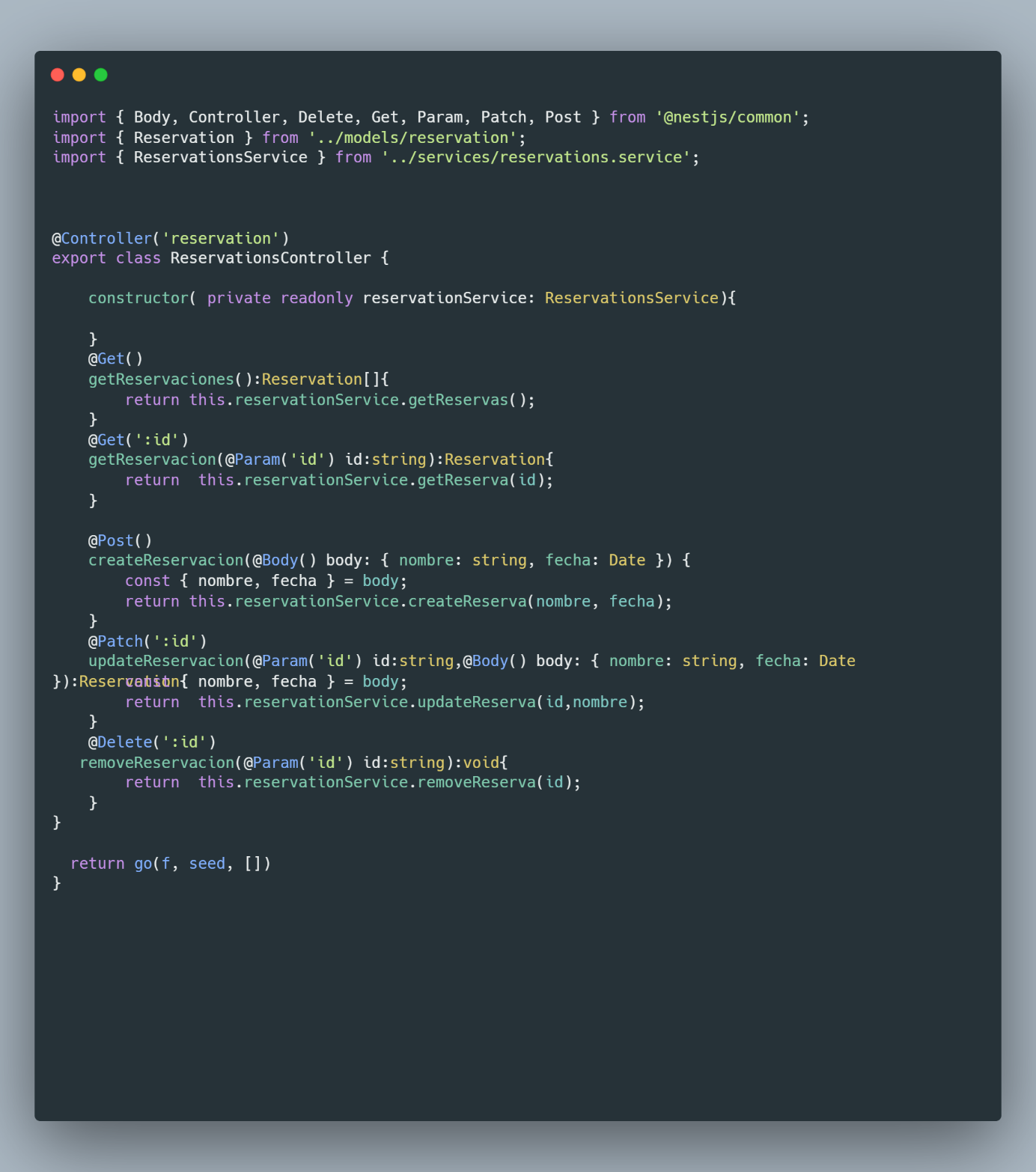
Ilustración Código Clase ReservatiopsService



Fuente: Elaboración propia

### Clase ReservationController

Ilustración Codigo Clase ReservationController



Fuente: Elaboración propia

## Conclusiones:

El Diseño Orientado al Dominio (DDD) es un enfoque del desarrollo de software que subraya la importancia del modelo de dominio en el diseño y la implementación de sistemas de software. El DDD puede ayudar a crear sistemas de software que reflejen con precisión el dominio del problema y sean fáciles de entender, mantener y evolucionar con el tiempo. DDD hace hincapié en el uso de patrones y mejores prácticas para garantizar que el modelo de dominio está bien diseñado y bien implementado. La DDD puede integrarse con otros aspectos del desarrollo de software, como las metodologías de desarrollo ágil, el desarrollo basado en pruebas y la integración continua. Existen varias herramientas y tecnologías que pueden utilizarse para apoyar el DDD, como los lenguajes específicos de dominio, los marcos de mapeo objeto-relacional, los marcos de aprovisionamiento de eventos y las herramientas de orquestación y contenedorización. Entre los ejemplos reales de aplicación con éxito de la DDD figuran las plataformas de comercio electrónico, los sistemas de comercio financiero y los sistemas de gestión sanitaria.

## Bibliografía:

* [Qué es Domain Driven Design (DDD), ventajas y proceso básico (hubspot.es)](https://blog.hubspot.es/website/que-es-ddd)
* [DDD: ¿Qué es y cómo se aplica? - TECNOVA](https://www.tecnova.cl/2021/06/23/ddd-domain-driven-design/)
* [Introducción a DDD (Domain-Driven Design) - Alberto Moreno’s .NET blog (arbems.com)](https://arbems.com/introduccion-a-ddd-domain-driven-design/)
* [TFM\_Murillo\_Paredes\_2021.pdf (uah.es)](https://ebuah.uah.es/dspace/bitstream/handle/10017/49932/TFM_Murillo_Paredes_2021.pdf?isAllowed=y&sequence=1)
* Evans, E. (2004). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley.
* Vernon, V. (2013). Implementing Domain-Driven Design. Addison-Wesl
* [dddcommunity.org | Domain Driven Design Community](https://www.dddcommunity.org/)
* [Domain Driven Design Quickly (infoq.com)](https://www.infoq.com/minibooks/domain-driven-design-quickly/)
* [Diseño guiado por el dominio - Wikipedia, la enciclopedia libre](https://es.wikipedia.org/wiki/Dise%C3%B1o_guiado_por_el_dominio)
* [Los principios del diseño basado en dominios (DDD) y cómo guían a los arquitectos de software | Rootstack](https://rootstack.com/es/blog/los-principios-del-diseno-basado-en-dominios-ddd-y-como-guian-los-arquitectos-de-software)
* [Arquitectura DDD para el desarrollo de software – Tekudo Tech (wordpress.com)](https://tekudo.wordpress.com/2023/07/10/arquitectura-ddd-para-el-desarrollo-de-software/)