UNIVERSIDAD NACIONAL DE JOSÉ C PAZ

PROYECTO HOTEL

CARRERA: LGTI

MATERIA: LABORATORIO DE SOFTWARE

COMISIÓN: C1

PROFESOR: DANIEL FERNANDEZ

GRUPO: EDUARDO ARIZZA
GONZALO ARIZZA
JULIA AVALOS
OMAR BAZAR

1 Índice

1. PROYECTO DE ARQUITECTURA LIMPIA DDD

1.1 Presentación del Proyecto Hotel:

El proyecto Hotel es una aplicación de gestión de clientes y reservas para un hotel, diseñada utilizando la arquitectura de Diseño Dirigido por el Dominio (DDD). La aplicación se divide en varias capas bien definidas, cada una con responsabilidades específicas, lo que facilita la mantenibilidad y escalabilidad del sistema.

1.2 Objetivos:

- Desarrollar una aplicación de gestión de clientes y reservas para un hotel que permita una administración eficiente de datos.
- Implementar la arquitectura DDD para lograr una separación clara de responsabilidades y facilitar la mantenibilidad y escalabilidad del sistema.
- Asegurar la mantenibilidad del código mediante la aplicación de los principios SOLID.

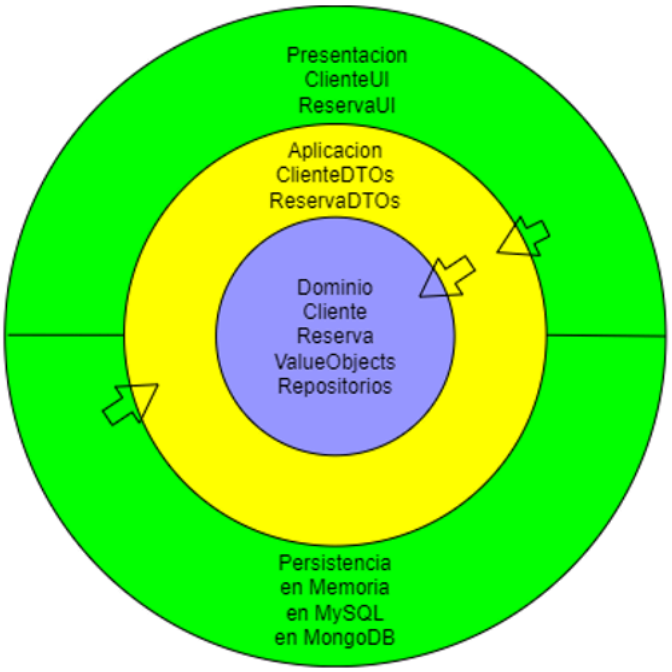
1.3 Requisitos Funcionales:

- Gestión de clientes: crear, editar, eliminar y consultar información de clientes.
- Gestión de reservas: crear, editar, eliminar y consultar información de reservas.
- Interfaz de usuario: mostrar menús y opciones para gestionar clientes y reservas.
- Persistencia de datos: almacenar y recuperar información de clientes y reservas en una base de datos.

1.4 Requisitos no Funcionales

- Escalabilidad: El sistema debe ser capaz de manejar un incremento en el número de clientes y reservas sin una degradación significativa en el rendimiento.
- Mantenibilidad: El código debe ser fácil de entender, modificar y extender. Uso de principios SOLID para asegurar una arquitectura limpia.

1.5 DIAGRAMA ARQUITECTURA ONION



1.5.1 Breve definición de la arquitectura onion utilizada

Esta arquitectura representa un modelo de software llamado arquitectura en capas. La cual se encuentra conformada de la siguiente manera:

- Dominio: Es la capa central y contiene la lógica principal del negocio. Aquí se encuentran conceptos como “Cliente,” “Reserva,” “ValueObjects,” y “Repositorios.”

- Aplicación: Esta capa maneja la transferencia de datos entre las capas. Incluye “ClienteDTOs” y “ReservaDTOs.”

- Presentación: Es la capa exterior y se encarga de las interfaces de usuario (UI) para clientes y reservas.

- Persistencia: Representada por “Persistencia en Memoria,” “MySQL,” y “MongoDB,” esta capa almacena los datos.

2 LISTA DE REQUERIMIENTOS FUNCIONALES

3 REQUERIMIENTOS FUNCIONALES:

REQ_001- Crear Nuevo Cliente

Permitir al usuario ingresar los datos de un cliente (nombre, email, clave, fecha de nacimiento). Validar el ingreso de los datos ingresados por el usuario (por ejemplo, formato de fecha). Y, por último, almacenar la información del nuevo cliente en el repositorio seleccionado.

REQ_002 – Listar Clientes

Permitir al usuario obtener una lista de todos los clientes registrados, mostrando detalles como ID, nombre, email, clave y fecha de nacimiento.

REQ_003 – Borrar Cliente

Permitir al usuario borrar un cliente a través de su ID. Y poder manejar errores en caso de ID inválido o si el cliente no existe.

REQ_004- Actualizar Cliente

El sistema debe permitir la actualización de la información de un cliente existente en el sistema.

REQ_005 – Crear Nueva Reserva

Permite al usuario ingresar los datos de una reserva (ID de cliente, fecha de inicio, fecha de fin, precio), validando la entrada del usuario (por ejemplo, formato de fecha). Debe almacenar la información de la nueva reserva en el repositorio seleccionado.

REQ_006 – Listar Reservas

Permite al usuario obtener una lista de todas las reservas registradas, mostrando los detalles de la misma, como ID, ID de cliente, fecha de inicio, fecha de fin.

REQ_007 – Borrar Reserva

Permitir al usuario borrar una reserva a través de su ID. Debe manejar errores en caso de ID inválido o si la reserva no existe.

REQ_008 – Selección Base de Datos

Permitir al usuario seleccionar entre bases de datos en memoria, MySQL y MongoDB. Realizando las configuraciones de los repositorios correspondientes según la base de datos seleccionada.

5 CASOS DE USO

<table>
<tr><td>Título:</td><td>01. CREAR NUEVO CLIENTE</td></tr>
<tr><td>Actores involucrados:</td><td>Usuario</td></tr>
<tr><td colspan="2">Objetivo:</td></tr>
<tr><td colspan="2">Registrar un nuevo cliente en el sistema de manera correcta.</td></tr>
<tr><td colspan="2">Precondición:</td></tr>
<tr><td colspan="2">El sistema debe estar en funcionamiento y el repositorio debe estar seleccionado.</td></tr>
<tr><td colspan="2">Postcondición:</td></tr>
<tr><td colspan="2">Se creó un nuevo cliente y los datos son guardados en el repositorio.</td></tr>
<tr><td>Escenario Principal</td><td>Flujo Alternativo</td></tr>
<tr><td>1. El usuario ingresa al sistema
2. Selecciona la opción "Ingresar nuevo cliente" en el menú.
3. El sistema solicita los datos del cliente (nombre, email, clave, fecha de nacimiento).
4. El usuario ingresa los datos solicitados.
5. El sistema valida los datos ingresados.
6. El sistema guarda los datos del nuevo cliente en el repositorio.
7. El sistema confirma al usuario que el cliente fue ingresado éxito.
8. Fin del caso de uso</td><td>3.1.1 Si los datos ingresados no son válidos:
3.1.2 El sistema muestra un mensaje de error y solicita la corrección de los datos.
3.1.3 El usuario corrige los datos y vuelve al paso 4 del escenario principal.</td></tr>
</table>

Título:	02. LISTAR CLIENTES
Actores involucrados:	Usuario
Objetivo:	
Mostrar la lista de todos los clientes registrados en el sistema.	
Precondición:	
El sistema debe estar en funcionamiento, el repositorio debe estar seleccionado y debe haber clientes registrados previamente.	
Postcondición:	
La lista de clientes se muestra al usuario.	

Escenario Principal	**Flujo Alternativo**
1. El usuario selecciona la opción "Listar clientes" en el menú 2. El sistema recupera la lista de clientes del repositorio. 3. El sistema muestra la lista de clientes al usuario. 4. Fin del caso de uso	2.1 Si no hay clientes en el repositorio: 2.1.1 El sistema muestra un mensaje indicando que no hay clientes para listar. 2.1.2 Fin de caso de uso.

Título:	03. BORRAR CLIENTE
Actores involucrados:	Usuario
Objetivo:	
Permitir al usuario eliminar un cliente existente del sistema.	
Precondición:	
El sistema debe estar en funcionamiento, el repositorio debe estar seleccionado y el cliente debe estar persistido en el sistema	
Postcondición:	
El cliente es eliminado del repositorio o se informa al usuario si no fue posible borrar al cliente.	

Escenario Principal	**Flujo Alternativo**
1. El usuario ingresa al sistema 2. El usuario selecciona la opción "Borrar cliente" en el menú 3. El sistema solicita el ID del cliente a borrar. 4. El usuario ingresa el ID del cliente. 5. El sistema valida el ID ingresado. 6. El sistema intenta borrar el cliente del repositorio. 7. El sistema confirma al usuario que el cliente fue borrado con éxito o muestra un mensaje de error si no fue posible. 8. Fin del caso de uso	4.1 Si el ID ingresado no es válido: 4.1.1 El sistema muestra un mensaje de error y solicita la corrección del ID. 4.1.2 El usuario corrige el ID y vuelve al paso 4 del escenario Principal. 5.1 Si no se puede borrar el cliente (por ejemplo, porque no existe): 5.1.1 El sistema muestra un mensaje de error indicando el problema. 5.1.2 Fin del caso de uso.

Título:	04. ACTUALIZAR CLIENTE
Actores involucrados:	Usuario
Objetivo:	
Actualizar la información de un cliente en el sistema con nuevos datos proporcionados por el usuario.	
Precondición:	
El usuario debe conocer el ID del cliente que desea actualizar. El cliente debe existir en el sistema.	
Postcondición:	
La información del cliente es actualizada en el sistema si todos los datos proporcionados son válidos.	

Escenario Principal	**Flujo Alternativo**
1. El usuario selecciona la opción para actualizar un cliente. 2. El sistema solicita el ID del cliente a actualizar. 3. El usuario ingresa el ID del cliente. 4. El sistema verifica si el ID es válido. 5. El sistema solicita el nuevo nombre del cliente. 6. El usuario ingresa el nuevo nombre. 7. El sistema solicita el nuevo email del cliente. 8. El usuario ingresa el nuevo email. 9. El sistema solicita la nueva clave del cliente. 10. El usuario ingresa la nueva clave. 11. El sistema solicita la nueva fecha de nacimiento del cliente 12. El usuario ingresa la nueva fecha de nacimiento. 13. El sistema verifica si todos los datos son válidos. 14. El sistema actualiza la información del cliente. 15. El sistema informa al usuario que la actualización fue exitosa 16. Fin del caso de uso.	4.1 ID de Cliente Inválido: 4.1.1 Si el ID del cliente ingresado no es válido, el sistema informa al usuario y no continúa con la actualización. 4.1.2 Fin del caso de uso 13.1 Datos Nulos: 13.1.1 Si alguno de los datos ingresados es nulo (nombre, email o clave), el sistema informa al usuario que no pueden ser nulos y no realiza la actualización. 13.1.2 Fin del caso de uso 13.2 Fecha de Nacimiento Inválida: 13.2.1 Si la fecha de nacimiento ingresada no es válida, el sistema informa al usuario y no realiza la actualización. 13.2.2 Fin del caso de uso 14.1 Error durante la Actualización: 14.1.1 Si ocurre un error durante la actualización (por ejemplo, una excepción), el sistema informa al usuario del error específico y no realiza la actualización. 14.1.2 fin del caso de uso

Título:	05. CREAR NUEVA RESERVA
Actores involucrados:	Usuario
Objetivo:	
Registrar una nueva reserva en el sistema de manera correcta.	
Precondición:	
El sistema debe estar en funcionamiento, el repositorio debe estar seleccionado, y como mínimo debe haber un registro de cliente en la base de datos	
Postcondición:	
Los datos de la nueva reserva son guardados en el repositorio.	
Escenario Principal	**Flujo Alternativo**
1. El usuario selecciona la opción "Crear nueva reserva" en el menú. 2. El sistema solicita los datos de la reserva (ID de cliente, fecha de inicio, fecha de fin, precio). 3. El usuario ingresa los datos solicitados. 4. El sistema valida los datos ingresados. 5. El sistema guarda los datos de la nueva reserva en el repositorio. 6. El sistema confirma al usuario que la reserva fue creada con éxito.	4.1 Si los datos ingresados no son válidos: 4.1.1 El sistema muestra un mensaje de error y solicita la corrección de los datos. 4.1.2 El usuario corrige los datos y vuelve al paso 4 del escenario principal.

Título:	06. LISTAR RESERVAS
Actores involucrados:	Usuario
Objetivo:	
Permitir al usuario obtener una lista de todas las reservas registradas en el sistema.	
Precondición:	
El sistema debe estar en funcionamiento, el repositorio debe estar seleccionado, previamente se debe haber creado y persistido menos una, reserva	
Postcondición:	
La lista de reservas se muestra al usuario.	
Escenario Principal	**Flujo Alternativo**
1. El usuario selecciona la opción "Listar reservas" en el menú. 2. El sistema recupera la lista de reservas del repositorio. 3. El sistema muestra la lista de reservas al usuario. 4. Fin del caso de uso.	2.1 Si no hay reservas en el repositorio: 2.1.1 El sistema muestra un mensaje indicando que no hay reservas para listar. 2.1.2 Fin del caso de uso.

Título:	07. BORRAR RESERVA
Actores involucrados:	Usuario
Objetivo:	
Permitir al usuario eliminar una reserva existente del sistema.	
Precondición:	
El sistema debe estar en funcionamiento, el repositorio debe estar seleccionado, previamente se debe haber creado y persistido, al menos una, reserva	
Postcondición:	
La reserva es eliminada de la base de datos	
Escenario Principal	**Flujo Alternativo**
1. El usuario selecciona la opción "Borrar reserva" en el menú. 2. El sistema solicita el ID de la reserva a borrar. 3. El usuario ingresa el ID de la reserva. 4. El sistema valida el ID ingresado. 5. El sistema intenta borrar la reserva del repositorio. 6. El sistema confirma al usuario que la reserva fue borrada con éxito. 7. Fin del caso de uso.	4.1 Si el ID ingresado no es válido: 4.1.1 El sistema muestra un mensaje de error y solicita la corrección del ID. 4.1.2 El usuario corrige el ID y vuelve al paso 4 del escenario principal. 5.1 Si no se puede borrar la reserva (por ejemplo, porque no existe): 5.1.1 El sistema muestra un mensaje de error indicando el problema. 5.1.2 Fin del caso de uso.

Título:	08. SELECCIÓN BASE DE DATOS
Actores involucrados:	Usuario
Objetivo:	
Seleccionar y configurar la base de datos que utilizará el sistema	
Precondición:	
El sistema debe estar en funcionamiento.	
Postcondición:	
La base de datos seleccionada está configurada y lista para ser utilizada por el sistema.	
Escenario Principal	**Flujo Alternativo**
1. El usuario selecciona el tipo de base de datos en el menú principal. 2. El sistema configura los repositorios correspondientes según la selección del usuario. 3. El sistema confirma al usuario que la base de datos ha sido seleccionada y configurada con éxito. 4. Fin del caso de uso	2.1 Si ocurre un error al configurar los repositorios: 2.1.1 El sistema muestra un mensaje de error indicando el problema. 2.1.2 Fin del caso de uso.

6. PRUEBA UNITARIA EMAIL TEST

Importancia de la Prueba de Validación de Emails en Despliegue, Implementación y Control de Calidad

Despliegue

- **Confiabilidad:** Esta prueba garantiza que la clase ***Email*** en el sistema maneje correctamente los emails válidos e inválidos antes de que el sistema sea desplegado en producción. Esto es crucial porque asegura que la validación de emails, una función esencial, no fallará cuando los usuarios reales utilicen el sistema.
- **Prevención de Errores:** Al ejecutar esta prueba unitaria antes del despliegue, se puede identificar y corregir errores relacionados con la validación de emails. Esto reduce significativamente la posibilidad de que estos errores lleguen a afectar a los usuarios finales.

Implementación

- **Desarrollo Guiado por Pruebas (TDD):** este tipo de pruebas son una parte clave del TDD. Se define qué comportamiento se espera (un email válido no lanza una excepción, un email inválido lanza una excepción) y luego se implementa el código para cumplir con estos requisitos. Esto asegura que cada parte del código cumpla con las especificaciones y funcione correctamente desde el principio.
- **Documentación del Comportamiento:** Esta prueba documenta claramente cómo debe comportarse la ***Email*** clase . Cualquier desarrollador que trabaje en el proyecto puede entender rápidamente los requisitos de validación de emails simplemente revisando esta prueba.
- **Facilidad de Refactorización:** Si en algún momento se necesitará cambiar cómo se valida un email (por ejemplo, para añadir nuevas reglas de validación), este tipo de pruebas permiten hacerlo con confianza. Se puede modificar la implementación sabiendo que las pruebas alertarán si uno introduce errores.

Control de Calidad

- **Verificación Continua:** En un entorno de integración continua, este tipo de pruebas se ejecutan automáticamente cada vez que se realiza un cambio en el código. Esto garantiza que cualquier modificación no rompa la funcionalidad existente de validación de emails.
- **Reducción de Errores en Producción:** Validar correctamente los emails es crucial para la calidad general del sistema. Usuarios ingresando emails inválidos podrían causar problemas en la comunicación y en el flujo de trabajo. Esta prueba ayuda a asegurar que solo se aceptan emails válidos, mejorando así la experiencia del usuario.
- **Mantenimiento de Estándares de Calidad:** Al asegurar que las entradas de email sean validadas correctamente, esta prueba ayuda a mantener un alto estándar de calidad en el sistema. Que es fundamental para la satisfacción del usuario final.