Programación Dos.

Programación OOP con C# en Microsoft Visual Studio y SQL

Proyecto N° 2

Gestión para el “Hotel del Estrella del Mar”.

**Profesor:** Carlos Rodríguez.

**Alumno:** Paula Camacho.

Noviembre 2024

Indice:

[2 Declaración de autoría. 3](#_Toc184585314)

[3 Introducción: 4](#_Toc184585315)

[4 Abstract: 5](#_Toc184585316)

[5 Diccionario de clases: 7](#_Toc184585317)

[6 Funciones que destaco de la Clase Program: 12](#_Toc184585318)

[7 Metodos de la Clase Hotel a destacar: 14](#_Toc184585319)

[8 Herencia: 16](#_Toc184585320)

[9 Diagrama de clases: 18](#_Toc184585321)

[10 Diagrama de clases con metodos: 19](#_Toc184585322)

[11 Sistema funcionando en consola: 20](#_Toc184585323)

[12 Conexión a base de datos: 22](#_Toc184585324)

[13 Entity framework core: 23](#_Toc184585325)

[14 Data annotations: 24](#_Toc184585326)

[15 Cardinalidad: 27](#_Toc184585327)

[16 Cronograma de trabajo: 28](#_Toc184585328)

[17 Bibliografía: 29](#_Toc184585329)

# Declaración de autoría.

Yo, Paula Camacho, declaro que el trabajo que se presenta es de mi propia mano. Puedo asegurar que la obra fue producida mientras cursaba la materia Programación Dos con el profesor Carlos Rodríguez. He tenido en cuenta las clases dictadas por el Prof. Carlos Rodríguez, quien además proporcionó el material. Cuando he consultado el trabajo publicado por otros, lo he atribuido con claridad, y cuando he citado obras de otros, he indicado las fuentes. Con excepción de estas citas, la obra es enteramente mía. En la obra, he acusado recibo de las ayudas recibidas y cuando la obra se basa en trabajo realizado conjuntamente con otros, he explicado claramente qué fue construido por otros, y qué fue contribuido por mí. Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Firma:

Paula Camacho

# Introducción:

El presente documento describe el desarrollo del Sistema de Gestión de Reservas del Hotel Estrella del Mar, una aplicación de consola creada en C# (.NET 6.0) que tiene como objetivo la gestión eficiente y automatizada de las reservas, consultas de habitaciones y pagos dentro del hotel. El sistema está diseñado para permitir tanto a los usuarios como al personal administrativo llevar a cabo operaciones clave, como la consulta de información sobre el hotel, la realización de reservas, la modificación o cancelación de las mismas, y la gestión de pagos relacionados con las estadías.

El Hotel Estrella del Mar se encuentra en una ciudad costera turística, ofreciendo una experiencia única para sus huéspedes mediante instalaciones modernas y una variedad de opciones de habitaciones adaptadas a diversas necesidades. Con este sistema, se busca mejorar la eficiencia operativa del hotel, optimizando la gestión de reservas y la experiencia del cliente, asegurando que cada huésped tenga un acceso rápido y sencillo a los servicios que se le ofrecen, además de proporcionar al personal del hotel herramientas efectivas para la administración de las operaciones diarias.

A lo largo de este documento se detalla la estructura del proyecto, sus componentes principales, y los procedimientos de implementación de cada funcionalidad clave, como la creación y gestión de reservas, la modificación de datos de los huéspedes, y la generación de reportes de las actividades realizadas en el hotel.

# Abstract:

El Hotel Estrella del Mar enfrenta el desafío de gestionar eficientemente las reservas, las habitaciones disponibles y los pagos de sus huéspedes de manera manual, lo que puede resultar en procesos lentos, errores humanos y dificultad para acceder rápidamente a la información crucial. Actualmente, el sistema de gestión no permite realizar reservas de forma automatizada, lo que genera confusión y posibles conflictos de disponibilidad de habitaciones, como reservas duplicadas o en fechas no disponibles. Además, el personal del hotel debe llevar un control manual de las reservas, lo que aumenta el riesgo de errores y dificulta la gestión de la ocupación del hotel.

El sistema propuesto busca automatizar y mejorar estos procesos, permitiendo a los usuarios realizar reservas de forma rápida y sencilla a través de una interfaz de consola, gestionando automáticamente la disponibilidad de las habitaciones, evitando duplicados y asegurando que se respeten las fechas de reserva. Además, el sistema permitirá al personal administrativo realizar modificaciones, cancelaciones y consultas de reservas de manera eficiente y sin riesgo de error.

Tecnología Utilizada:

La solución está desarrollada utilizando C# como lenguaje de programación, en la IDE Visual Studio, sobre la plataforma .NET 6.0, aprovechando la facilidad de mantenimiento y escalabilidad. La aplicación es de tipo consola, lo que permite su implementación rápida y fácil, además de ser una opción ligera para su uso en el entorno de un hotel pequeño o mediano.

El sistema utiliza una arquitectura orientada a objetos (POO), lo que facilita la organización del código, la reutilización de componentes y el mantenimiento a largo plazo. Además, se implementa el uso de colecciones de datos como listas, lo que permite gestionar de forma eficiente las reservas y las habitaciones del hotel.

En la aplicación de consola, he creado un menú que presenta al usuario una lista de opciones para interactuar con el sistema. A través de este menú, se pueden visualizar los métodos y la información que el usuario puede agregar, tales como gestionar reservas, consultar disponibilidad de habitaciones y realizar pagos.

Para simplificar las pruebas, he incluido un usuario de prueba predefinido con las siguientes credenciales:

- Email: Juancho@gmail.com

- Contraseña: UwU

Este usuario harcodeado permite acceder al sistema y probar las funcionalidades sin necesidad de registrarse, facilitando el proceso de desarrollo y demostración del sistema.

# Diccionario de clases:

Tabla de atributos de la Clase Usuario:

|  |  |  |
| --- | --- | --- |
| Atributo | Tipo de dato | Descripción |
| Email | string | Email del usuario |
| Contrasenia | object | Contraseña del usuario |
| Usuario | Usuario | Lista de usuario |
|  |  |  |

Tabla de métodos de la Clase Usuario:

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Devuelve | Recibe(Parametros) | Descripción |
| ToString | Nada | Niguno | Devuelve la instancia en string |

Tabla atributos Clase Pago

|  |  |  |
| --- | --- | --- |
| Atributo | Tipo de dato | Descripcion |
| idPago | Int | Id del pago |
| fechaPago | DateTime | Fecha del pago |
| Monto | decimal | Cantidad a pagar |
| tipoPago | string | Qué clase de pgo |

Tabla Clase PagoConTarjeta:

|  |  |  |
| --- | --- | --- |
| Atributo | Tipo de dato | Descripcion |
| NumeroTarjeta | int | Número de la tarjeta con la que se paga |
| TipoTarjeta | string | Tipo de tarjeta, si es Credito o Debito |

Tabla atributos Pago Efectivo:

|  |  |  |
| --- | --- | --- |
| Atributo | Tipo de dato | Descripcion |
| CambioEntregado | decimal | Vuelto del pago Efectivo |

Tabla Clase Huesped:

|  |  |  |
| --- | --- | --- |
| Atributo | Tipo de dato | Descripcion |
| idHuesped | Int | Id del huésped |
| nombreHuesped | string | Nombre del huésped |
| apellidoHuesped | string | Apellido del huésped |
| tipoDocHuesped | string | Tipo documento del huésped |
| documentoNHues | double | Numero de documento |
| fechaNacimientoHues | DateTime | Fecha de nacimiento del huésped |
| numeroCelHues | int | Celular del huésped |
| emailHues | sting | Mail del huésped |
| paisHues | strin | País de ubicación del huésped |
| Usuario | List<Usuario> | Lista del tipo Usuario, refiriendo al Ususario |

Tabla métodos Huesped:

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Devuelve | Recibe(Parametros) | Descripción |
| ToString | String | Nada | Devuelve una string del objeto seleccionado |

Tabla Clase Habitación:

|  |  |  |
| --- | --- | --- |
| Atributo | Tipo de dato | Descripcion |
| idNumeroHabitacion | Int | Id de la habitacion |
| tipoHabitacion | string | Tipo dehabitacion |
| cantidadHuesped | Int | Cantidad de huespedes |
| precioDiario | Int | precio diario de la habitacion |
| disponible | bool | Si esta disponible o no, true or false |

Tabla Metodos Habitacion:

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Devuelve | Recibe(Parametros) | Descripción |
| ToString | string | No | Devuelve una lista de habitaciones en string |

Tabla Clase Hotel:

|  |  |  |
| --- | --- | --- |
| Atributo | Tipo de dato | Descripcion |
| hotelId | Int | Id del hotel |
| hotelName | string | Nombre del hotel |
| ubicHotel | string | Ubicación del hotel |
| categoria | int | Categoria del hotel |
| servicio | string | Servicio del hotel |
| habitaciones | List<Habitacion> | Lista del tipo Habitacion(clase) |
| personas | List<Huesped> | Lista del tipo Huesped(clase) |
| reservas | List<Reserva> | Lista del tipo Reserva(clase) |

Tabla Metodos Hotel:

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Devuelve | Recibe(Parametros) | Descripción |
| AgregarHabitacion | void | Habitacion habitacion | Agrega una instancia habitacion a la lista habitacion |
| ListarHabitaciones | void | No | Recorre cada habitación de la lista habistaciones |
| ListarReseras | void | Int id | Busca las reservas especificadas, en una lista de reservas y muestra el id en consola |
| AgregarHuesped | void | No |  |
| ListarHuesped | void | No | Agrega huésped a una lista |
| ListarHabitacionesDisponibles | List | No | si está disponible la habitación la lista |
| ToString | string | No | Da la string del objeto |

Tabla Metodo GestionDeHuesped:

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Devuelve | Recibe(Parametros) | Descripción |
| CargarHuesped | Lista | No | Lista de Huespedes precargados |

Tabla Metodos GestionDeUsusario:

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Devuelve | Recibe(Parametros) | Descripción |
| CargarUsuario | Lista | No | Tiene una lista de tres usuarios precargados, en la cual se usan dos |

Tabla Metodos GestionDeHabitaciones:

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Devuelve | Recibe(Parametros) | Descripción |
| CargarHabitaciones | Lista habitaciones precargadas | No | Lista habitaciones precargadas con un loop foreach y ternarios |

Tabla Metodos PrecargaReserva:

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Devuelve | Recibe(Parametros) | Descripción |
| PrecargaReserva | reservas | No | Es una lista de Reservas precargadas |

# Funciones que destaco de la Clase Program:

PrecargarUsuario()

- Descripción: Carga y selecciona un usuario específico de una lista de usuarios precargados.

- Función en el sistema: Utilizada para simular un usuario ya registrado que inicia sesión en el sistema.

PrecargarReserva()

-Descripción: Utiliza una lista de usuarios y habitaciones precargadas para crear y agregar reservas iniciales al hotel.

- Función en el sistema: Facilita la carga de reservas predeterminadas al sistema para realizar pruebas o inicializar datos sin la necesidad de ingresarlos manualmente cada vez.

IngresarNuevaReserva()

- Descripción: Solicita al usuario seleccionar una habitación y las fechas de entrada y salida para crear una nueva reserva. La función verifica que la duración no exceda 30 días y llama a métodos para listar habitaciones disponibles y agregar la reserva.

- Función en el sistema: Permite al usuario hacer reservas personalizadas, manejando tanto la selección de habitaciones como la validación de fechas, asegurando que la duración de las reservas sea adecuada.

ListarHabitacionesDisponibles()

- Descripción: Muestra todas las habitaciones disponibles en el hotel al usuario.

- Función en el sistema: Proporciona una lista de opciones de alojamiento, permitiendo que los usuarios seleccionen una habitación específica al reservar.

. EliminarReserva()

- Descripción: Permite al usuario eliminar una reserva existente, siempre y cuando la fecha de inicio de la reserva sea posterior a la fecha actual.

- Función en el sistema: Ofrece una funcionalidad de gestión de reservas al permitir cancelarlas si aún no han comenzado, mejorando la flexibilidad y control del cliente sobre su reserva.

AltaHabitacion()

- Descripción: Solicita datos para una nueva habitación y la agrega al inventario del hotel.

- Función en el sistema: Facilita la expansión de habitaciones disponibles, permitiendo al administrador del hotel agregar habitaciones según sea necesario.

AltaHuesped()

- Descripción: Solicita y registra la información de un nuevo huésped en el sistema.

.

BuscarHabitacionPorId(int id)

-Descripción: Busca una habitación específica por su ID en la lista de habitaciones del hotel y la retorna.

- Función en el sistema: Ayuda a ubicar y seleccionar una habitación específica, lo cual es esencial para procesar reservas y verificar la disponibilidad de habitaciones exactas.

IdReserva()

- Descripción: Genera un nuevo ID para cada reserva, incrementándolo en base a la cantidad actual de reservas.

- Función en el sistema: Asegura que cada reserva tenga un identificador único, evitando conflictos y manteniendo el sistema de reservas organizado.

# Metodos de la Clase Hotel a destacar:

AgregarHabitacion(Habitacion habitacion)

Descripción: Añade una nueva habitación a la lista de habitaciones del hotel.

ListarHabitaciones()

Descripción: Muestra la información de todas las habitaciones en el hotel.

Función en el sistema: Ofrece una vista general de las habitaciones, útil tanto para los administradores como para los clientes que consultan opciones.

ListarHabitacionesDisponibles()

Descripción: Devuelve una lista de habitaciones que están actualmente disponibles para reservar.

Función en el sistema: Permite identificar fácilmente las habitaciones que pueden ser asignadas a nuevas reservas, mejorando la eficiencia en la gestión de disponibilidad.

AgregarHuesped(Huesped huesped)

Descripción: Añade un nuevo huésped a la lista de personas en el sistema del hotel.

ListarHuesped()

Descripción: Muestra la información de todos los huéspedes registrados en el hotel.

Función en el sistema: Ofrece una lista completa de los clientes actuales y pasados, útil para consultas o reportes de la gestión hotelera.

AgregarReserva(Reserva reserva)

Descripción: Agrega una nueva reserva a la lista de reservas del hotel.

Función en el sistema: Permite registrar reservas de clientes, facilitando la gestión del historial de reservas y la asignación de habitaciones.

ListarReservas(int id)

Descripción: Muestra información sobre una reserva específica, utilizando su ID.

Función en el sistema: Proporciona confirmación visual de que una reserva se ha registrado correctamente y permite consultar la información de la reserva.

ToString()

Descripción: Genera una cadena de texto con la información del hotel, incluyendo su nombre, ubicación, categoría y servicios ofrecidos.

Función en el sistema: Sirve para presentar una vista general del hotel, ya sea para mostrarla en la interfaz o en reportes, proporcionando una descripción completa y rápida del establecimiento

# Herencia:

Clase padre: Pago

Clases hijas: PagoEfectivo y PagoConTarjeta

En mi proyecto de gestión de hotel, he diseñado una estructura de clases para manejar los pagos realizados por los clientes. Para ello, defino una clase base llamada Pago, que almacena la información general de un pago, y luego creo clases derivadas que representan diferentes tipos de pago. Esto me permite reutilizar la lógica común y extenderla para casos específicos, siguiendo los principios de herencia en Programación Orientada a Objetos.

namespace ProyectoHotel

{

internal class Pago // Clase padre

{

public int idPago { get; set; }

public DateTime fechaPago { get; set; }

public decimal monto { get; set; }

public string tipoPago { get; set; }

public Pago(int idPago, int idReserva, DateTime fechaPago, decimal monto, string tipoPago)

{

this.idPago = idPago;

this.fechaPago = fechaPago;

this.monto = monto;

this.tipoPago = tipoPago;

}

}

}

namespace ProyectoHotel

{

internal class PagoConTarjeta : Pago //Clase hija

{

public int NumeroTarjeta { get; set; }

public string TipoTarjeta { get; set; } // Credito o Debito

// Constructor que llama al constructor base y agrega detalles específicos de la tarjeta

public PagoConTarjeta(int idPago, int idReserva, DateTime fechaPago, decimal monto, string tipoPago, int numeroTarjeta, string tipoTarjeta)

: base(idPago, idReserva, fechaPago, monto, tipoPago)

{

NumeroTarjeta = numeroTarjeta;

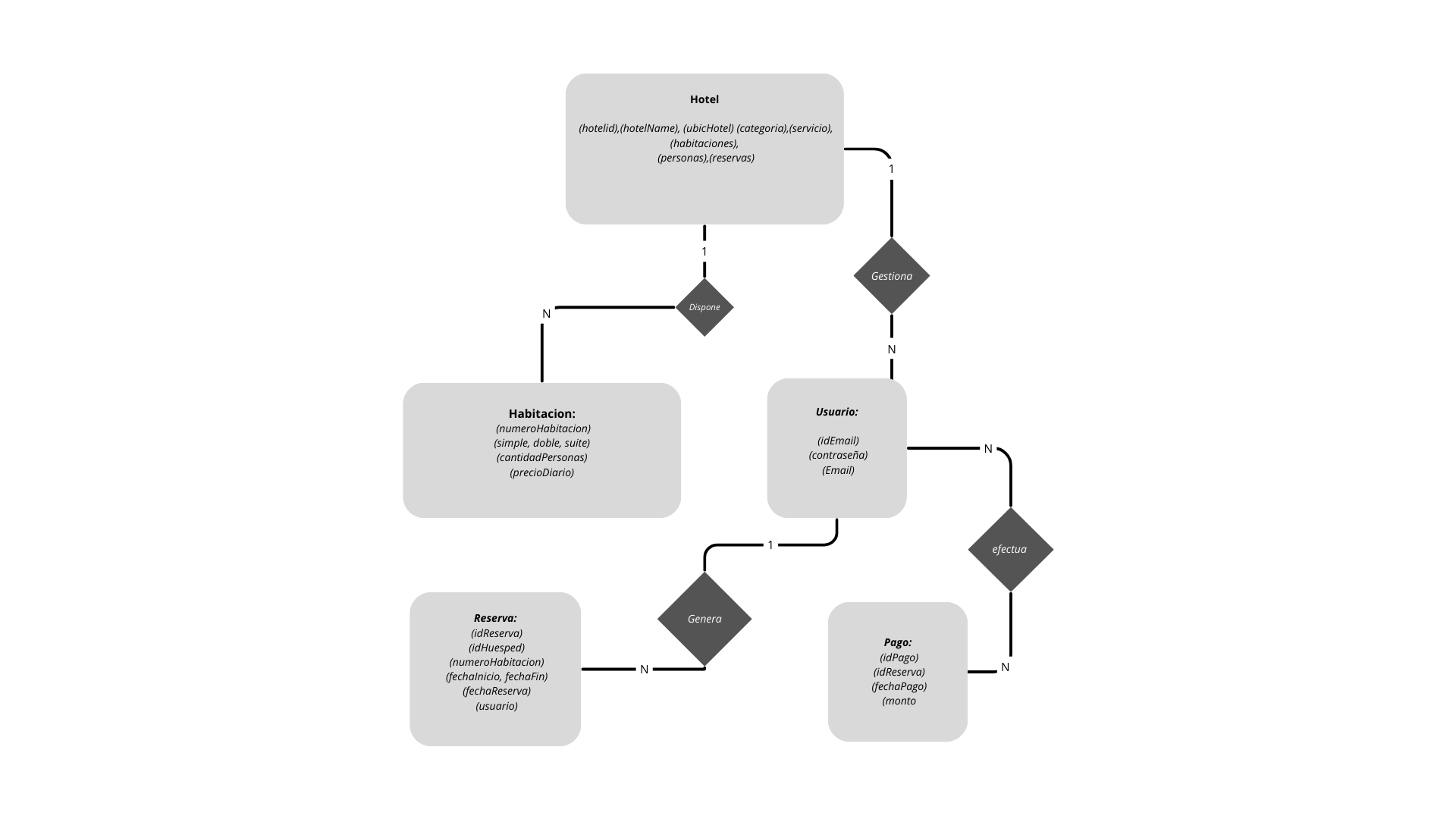
TipoTarjeta = tipoTarjeta;

}

}

}

# Diagrama de clases:



# C:\Users\PERSONAL\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Hotel (hotelid),(hotelName), (ubicHotel) (categoria),(servicio), (habitaciones), (personas),(reservas).pngDiagrama de clases con metodos:

# Sistema funcionando en consola:

Pasos a seguir para ingresar usuario y hacer reserva:

Consola:

“¡Bienvenido al hotel Estrella del Mar!

1 - Iniciar sesion

2 - Registrate

4 – Salir”

Si apretamos 1

“Ingresa tu mail:”

Ingresamos: [Juancho@gmail.com](mailto:Juancho@gmail.com)

“Ingrese su contrase;a:”

Ingresamos: UwU

Damos enter

En consola aparece

“1- Habitaciones disponibles

2 - Reservar Habitaciones”

Si apretamos 2 aparece en consola

“Habitacion: 1 Tipo: Simple Cantidad: 1, Precio 100

Habitacion: 2 Tipo: Simple Cantidad: 1, Precio 100

Habitacion: 3 Tipo: Simple Cantidad: 1, Precio 100

Habitacion: 4 Tipo: Simple Cantidad: 1, Precio 100

Habitacion: 5 Tipo: Simple Cantidad: 1, Precio 100

Habitacion: 6 Tipo: Doble Cantidad: 2, Precio 150

Habitacion: 7 Tipo: Doble Cantidad: 2, Precio 150

Habitacion: 8 Tipo: Doble Cantidad: 2, Precio 150

Habitacion: 9 Tipo: Doble Cantidad: 2, Precio 150

Habitacion: 10 Tipo: Doble Cantidad: 2, Precio 150

Habitacion: 11 Tipo: Suite Cantidad: 4, Precio 200

Habitacion: 12 Tipo: Suite Cantidad: 4, Precio 200

Habitacion: 13 Tipo: Suite Cantidad: 4, Precio 200

Habitacion: 14 Tipo: Suite Cantidad: 4, Precio 200

Habitacion: 15 Tipo: Suite Cantidad: 4, Precio 200

Nueva reserva:

Elija número de habitación:”

Ingreso el número de la habitación que deseo reservar, ejemplo, 15

“Elija fecha de entrada (D/M/A): “

Pongo la fecha 12/10/2024

“Elija fecha de salida (D/M/A):”

Escribo la fecha 20/10/2024

Al dar enter la consola me devuelve:

“reserva generada con exito, numero: 3

¿Deseas realizar otra reserva? (si/no):”

Si escribo no se termina el código

Si escribo si me lleva de nuevo al listado de habitaciones para reservar.

# Conexión a base de datos:

En mi proyecto, configuré la conexión a una base de datos SQL utilizando C#. Esto lo logré mediante la clase DbContext de Entity Framework Core, que actúa como un puente entre mi aplicación y la base de datos. Definí las entidades y las propiedades DbSet correspondientes para representar las tablas de la base de datos, y especifiqué la cadena de conexión en el método OnConfiguring o en el archivo de configuración de la aplicación.

Esta conexión me permite interactuar con la base de datos de manera eficiente para realizar operaciones como inserciones, actualizaciones, eliminaciones y consultas. La utilizo principalmente para gestionar las entidades clave de mi sistema, como habitaciones, usuarios y reservas, garantizando que los datos se almacenen y recuperen de forma consistente y segura. Además, al usar migraciones, puedo mantener sincronizada la estructura de la base de datos con el modelo de mi aplicación.

Código:

using Microsoft.EntityFrameworkCore;

namespace HotelProyectoWeb.Modelos

{

public class HotelDbContext : DbContext

{

public DbSet<Habitacion> Habitaciones { get; set; }

public DbSet<Usuario> Usuarios { get; set; }

public DbSet<Reserva> Reservas { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer("Server= DESKTOP-4JCRI14\\SQLEXPRESS;Database=AplicationRazor;Trusted\_Connection=True ;MultipleActiveResultSets=True;}");

}

}

}

# Entity framework core:

En este proyecto, conecté mi base de datos SQL utilizando Entity Framework Core como ORM para gestionar las entidades y realizar las operaciones sobre la base de datos. Para lograrlo, instalé y configuré los paquetes Entity Framework Core, Entity Framework Core Tools y Entity Framework Core SQL Server. Estos paquetes me permitieron crear y aplicar migraciones, definiendo la estructura de las tablas directamente desde las clases del modelo en mi aplicación. Con esta configuración, logré automatizar el proceso de creación y actualización de la base de datos de manera eficiente.

Paquetes utilizados:

* Entity Framework Core.
* Entity Framework Core Tools.
* Entity Framework Core SQL Server.

# Data annotations:

En el código se utilizan diversas **Data Annotations** para definir las propiedades y realizar validaciones en las entidades. Por ejemplo, se emplea [Key] para marcar propiedades como claves primarias, y [Required] para indicar que ciertos campos son obligatorios, permitiendo además personalizar los mensajes de error. Adicionalmente, se usan [StringLength] y [MaxLength] para restringir la longitud de cadenas, asegurando que no excedan un límite establecido, y [Range] para validar que los valores numéricos estén dentro de un rango permitido. También se incluye [DataType] para especificar el formato de los datos, como fechas, y [NotMapped] para indicar que ciertas propiedades no deben ser persistidas en la base de datos. Estas anotaciones garantizan integridad, coherencia y facilidad de mantenimiento en la aplicación.

Ejemplos con codigo :

* **[Key]**  
  Marca una propiedad como la clave principal de la entidad.  
  Ejemplo:  
  [Key]  
  public int hotelId { get; set; }

* **[Required]**  
  Indica que una propiedad es obligatoria. Se puede personalizar con un mensaje de error.  
  Ejemplo:  
  [Required(ErrorMessage = "El número de habitación es obligatorio.")]  
  public int numeroHabitacion { get; set; }
* **[StringLength]**  
  Establece la longitud máxima (y opcionalmente mínima) de una cadena de texto.  
  Ejemplo:  
  [StringLength(50, ErrorMessage = "El ID de usuario no puede exceder los 50 caracteres.")]  
  public string IdUsuario { get; set; }
* **[MaxLength]**  
  Define la longitud máxima permitida para cadenas o colecciones.  
  Ejemplo:  
  [MaxLength(20)]  
  public string tipoDocHuesped { get; set; }
* **[Range]**  
  Valida que un valor numérico se encuentre dentro de un rango específico.  
  Ejemplo:  
  [Range(1, 10000, ErrorMessage = "El precio diario debe ser mayor a 0 y menor o igual a 10,000.")]  
  public int precioDiario { get; set; }
* **[DataType]**  
  Especifica el formato de los datos, como fechas o correos electrónicos.  
  Ejemplo:  
  [DataType(DataType.Date, ErrorMessage = "La fecha de inicio debe ser una fecha válida.")]  
  public DateTime fechaInicio { get; set; }
* **[NotMapped]**  
  Indica que una propiedad no debe ser persistida en la base de datos.  
  Ejemplo:  
  [NotMapped]  
  public bool disponible { get; set; }
* **[EmailAddress]**  
  Valida que el valor sea un correo electrónico válido.  
  Ejemplo:  
  [EmailAddress(ErrorMessage = "Por favor, ingrese un correo electrónico válido.")]  
  public string Email { get; set; }
* **[RegularExpression]**  
  Define una expresión regular para validar el formato de los datos.  
  Ejemplo:  
    
  [RegularExpression(@"^\w+([.-]?\w+)@\w+([.-]?\w+)(.\w{2,3})+$", ErrorMessage = "Por favor ingrese su correo electrónico.")]  
  public string emailHues { get; set; }

# Cardinalidad:

En este proyecto, un hotel gestiona un número variable de habitaciones y usuarios. Cada usuario puede generar una o más reservas y efectuar uno o más pagos asociados a sus reservas. Las reservas están vinculadas tanto a un usuario como a una habitación, y los pagos se relacionan directamente con las reservas realizada

*Definición de Clases y Cardinalidades:*

Clase Hotel:

Tiene una relación **uno a muchos** con Habitacion (un hotel dispone de varias habitaciones).

También tiene una relación **uno a muchos** con Usuario (un hotel gestiona varios usuarios).

Clase Habitacion:

Relación **uno a muchos** con Reserva (una habitación puede estar asociada a múltiples reservas, en diferentes fechas).

Clase Usuario:

Relación **uno a muchos** con Reserva (un usuario puede generar varias reservas).

Relación **uno a muchos** con Pago (un usuario puede realizar múltiples pagos).

Clase Reserva:

Relación **muchos a uno** con Habitacion (una reserva se asocia a una única habitación).

Relación **muchos a uno** con Usuario (una reserva pertenece a un único usuario).

Relación **uno a muchos** con Pago (una reserva puede tener uno o más pagos asociados).

Clase Pago:

Relación **muchos a uno** con Reserva (un pago pertenece a una única reserva).

Relación **muchos a uno** con Usuario (un pago es realizado por un único usuario).

# Cronograma de trabajo:

|  |  |
| --- | --- |
| Actividad realizada: | Tiempo: |
| Diagrama de Capas. | 2 hs. |
| Control 1 Diagrama de Capas. | 2hs Entrega 15 de noviembre. |
| Actualizacion Diagrama de Capas | 1h. |
| Creación de Clases. | 2 hs. |
| Altas, bajas y modificaciones. | 10 hs Entrega 13 de noviembre. |
| Reservas | 16 hs |
| Menú | 4 hs |
| Diagrama de Capas UML | 2 hs |
| Testeo | 2hs. |
| Conexión BD | 2hs |
| Data annotations | 3hs |
| Cardinalidad | 30 min |
| Documentación | 10hs |

# Bibliografía:

1. Youtube.
2. StackOverflow.
3. ChatGpt.
4. Canvas.
5. FigJam.
6. Moodle CTC (material brindado por el profesor).
7. Clases de Zoom (material brindado por el profesor).