ITLA

AGOSTO
2025

PROYECTO FINAL

Documentación

Awilka Jerome Puente 2024-0114

Albin Lopez Sanchez 2024-1116

Programación 2 - Juan Rosario

CONTEXTO DEL PROYECTO

PROPÓSITO DEL SISTEMA

El presente proyecto consiste en un Sistema de Gestión de Citas, desarrollado como trabajo final de la materia Programación 2, impartida por el profesor Juan Rosario.
Para la realización del proyecto, nuestro equipo decidió basarse en el sistema de gestión de citas del INTRANT, adaptándolo y mejorándolo según las necesidades y objetivos del curso. Esto nos permitió crear un sistema funcional que incluye roles diferenciados para administradores y usuarios, permitiendo la gestión eficiente de horarios, reservas de citas y notificaciones.
El sistema busca ofrecer una experiencia clara y organizada, integrando buenas prácticas de desarrollo de software y adaptando la estructura del sistema real del INTRANT a un entorno educativo y controlado.

El propósito principal del Sistema de Gestión de Citas es proporcionar una herramienta eficiente y confiable para la administración y reserva de citas, tanto para los usuarios como para los administradores. Para los usuarios, el sistema permite consultar la disponibilidad de horarios, seleccionar la franja que mejor se ajuste a sus necesidades y recibir confirmaciones automáticas de sus reservas, garantizando una experiencia clara y organizada.
Para los administradores, el sistema ofrece un control centralizado sobre los horarios disponibles, la gestión de las citas programadas y la posibilidad de generar reportes o estadísticas que faciliten la toma de decisiones. Además, se asegura que la información se mantenga consistente y segura, evitando conflictos de horarios y errores manuales.

ARQUITECTURA

El sistema fue desarrollado utilizando la arquitectura Onion, lo que permitió un diseño escalable, limpio y mantenible, separando claramente las responsabilidades en diferentes capas.

CAPAS IMPLEMENTADAS

01 **DOMAIN**
contiene la lógica de negocio central, utilizando Domain-Driven Design (DDD), incluyendo Value Objects y el patrón Operation Result para manejar los resultados de las operaciones de manera consistente.

02 **APPLICATION**
orquesta la lógica de la aplicación, coordinando el flujo entre el dominio y la infraestructura.

03 **INFRASTRUCTURE**
manejo de la persistencia de datos, servicios externos y configuraciones del sistema.

04 **API**
Expone los servicios hacia el frontend o clientes externos.

PATRONES DE DISEÑO

STRATEGY PATTERN

ste patrón se utilizó para definir diferentes estrategias de envío de correos electrónicos dentro del sistema. Por ejemplo, existen estrategias distintas para enviar un correo cuando una cita es confirmada y otra diferente cuando una cita es cancelada.

FACTORY PATTERN

Este patrón complementa al Strategy, ya que se utiliza para crear los objetos correspondientes a cada estrategia de correo de manera flexible. La Factory permite instanciar el tipo de estrategia que se necesita según la situación, evitando acoplar directamente el código a clases concretas.

OPERATION RESULT PATTERN

Este patrón se implementó en la capa de dominio para gestionar los resultados de las operaciones de negocio de manera consistente. Cada acción realizada por el sistema, como crear una cita o cancelar una reserva, devuelve un objeto que contiene información sobre si la operación fue exitosa o fallida, junto con mensajes descriptivos y, en caso de error, detalles adicionales. Esto facilita la gestión de errores y la comunicación entre capas, asegurando que la capa de aplicación y la API reciban información clara sobre el estado de cada operación, sin necesidad de depender de excepciones o códigos de error dispersos.

DOMAIN-DRIVEN DESIGN

Y VALUE OBJECTS

Para el diseño de la capa de dominio se utilizó Domain-Driven Design (DDD), un enfoque que permite modelar el software de acuerdo a las reglas y conceptos del negocio. DDD ayuda a mantener la lógica de negocio aislada, coherente y organizada, evitando que detalles de infraestructura o presentación interfieran con las decisiones del dominio.

En nuestro proyecto, DDD se implementó principalmente en la entidad Usuario, donde se definieron atributos como Nombre, Cédula y Correo como Value Objects. Esto tiene varias ventajas

ENCAPSULAMIENTO DE REGLAS DE NEGOCIO

Cada Value Object valida y garantiza que los datos cumplan ciertas reglas antes de ser usados. Por ejemplo:

- Cedula solo acepta 11 dígitos numéricos.
- Correo debe cumplir un formato válido de email.
- Nombre no puede estar vacío, debe contener solo letras y espacios, y se normaliza automáticamente para mantener consistencia.

CLARIDAD EN EL MODELO DE DOMINIO

Al usar Value Objects, la entidad Usuario refleja conceptos del negocio de manera precisa. Por ejemplo, un usuario siempre tiene un Nombre, Correo y Cédula válidos, lo que asegura consistencia en toda la aplicación.

TECNOLOGÍAS Y LIBRERÍAS

BACKEND

El backend del sistema fue desarrollado utilizando C#, aprovechando la robustez y versatilidad del lenguaje para construir aplicaciones escalables y mantenibles. Se implementaron las siguientes tecnologías y librerías:

- Entity Framework Core: para la gestión de la base de datos y el mapeo objeto-relacional (ORM), permitiendo trabajar con entidades y colecciones directamente desde código C#, sin necesidad de escribir consultas SQL manuales en la mayoría de los casos.

- SQL (SQL Server): para almacenar toda la información del sistema, incluyendo usuarios, citas, roles y configuraciones de horarios.

- Mapster: utilizado para el mapeo de objetos, especialmente entre DTOs y entidades, facilitando la transferencia de datos entre la capa de dominio, aplicación y la API.

- JWT (JSON Web Tokens): implementado para seguridad y autenticación, garantizando que solo usuarios autorizados puedan acceder a ciertas rutas o funcionalidades del sistema.

- Secretos de usuario: se utilizan para guardar información sensible, como la cadena de conexión a la base de datos y la llave que proporciona Google para enviar correos vía SMTP, evitando que estos datos estén expuestos en el código fuente.

ENTIDADES Y RELACIONES

USUARIOS

Representa a las personas que pueden interactuar con el sistema, ya sea como usuarios regulares que reservan citas o como administradores que gestionan horarios y configuraciones.

Atributos principales:

- IdUsuario: identificador único.
- Nombre, Cedula, Correo: representados como Value Objects, garantizando consistencia y validación.
- FechaNacimiento, Telefono, Contrasena, Rol, Activo.

Relaciones:

- Un usuario puede tener muchas citas (ICollection<Cita>).
- Relacionado con la tabla cita mediante id_usuario.

CITA

Representa la reserva concreta realizada por un usuario.

Atributos principales:

- IdCita, IdUsuario, TurnoId, LugarId, ServicioId, Estado, FranjaId.

Relaciones:

- Usuarios → cita (1:N)
- Servicios → cita (1:N)
- Lugares → cita (1:N)
- ConfiguracionTurnos → FranjaHorario → cita

FRANJAHORARIO

Representa intervalos de tiempo específicos dentro de un turno donde un usuario puede reservar una cita.

SERVICIOS

Representa los servicios ofrecidos, cada uno con un nombre y precio.

Atributos principales:

- Id, Nombre, Precio.

Relaciones:

- Una cita está asociada a un servicio específico.

HORARIOS

Define los rangos de tiempo posibles para las citas, por ejemplo, de 8:00 AM a 4:00 PM.

Atributos principales:

- Id, HoraInicio, HoraFin, Descripcion.

Relaciones:

- Relacionado con configuracion_turnos mediante horarios_id.

CONFIGURACIÓN DE TURNOS

Define períodos de atención, duración de citas y número de estaciones disponibles.

Atributos principales:

- Id, FechaInicio, FechaFin, HorariosId, CantidadEstaciones, DuracionMinutos, AunAceptaCitas.

Relaciones:

- Vinculado a horarios mediante horarios_id.
- Relacionado con FranjaHorario para generar los intervalos de cita disponibles.
- Relacionado indirectamente con cita a través de FranjaId.

LUGAR

Define los lugares o estaciones donde se pueden atender las citas.

Atributos principales:

- Id, Nombre.

Relaciones:

- Cada cita se asigna a un lugar específico (cita.lugar_id).

FLUJO GENERAL ENTRE ENTIDADES

1. Usuario inicia sesión y accede al sistema.
2. Consulta los servicios disponibles.
3. Selecciona un turno activo (ConfiguracionTurnos) y revisa las franjas horarias generadas.
4. Escoge un lugar y una franja específica, y se crea la cita.
5. La cita queda vinculada a:
    a. Usuario (id_usuario)
    b. Servicio (servicio_id)
    c. Lugar (lugar_id)
    d. FranjaHorario (franja_id)
6. El estado de la cita se actualiza y se notifica al usuario mediante la estrategia de correo correspondiente.

MODELO ENTIDAD-RELACIÓN (MER)

- Un usuario puede tener muchas citas, pero cada cita pertenece a un solo usuario.
- Cada cita está asociada a un servicio, un lugar y una franja horaria específica.
- Las franjas horarias se generan a partir de una configuración de turno, que a su vez depende de un horario base.