Informe Técnico

AppGestionClinica; Herramientas de programación II

python-docx

i. U Pascual bravo

Informe Técnico – Sistema de Gestión de Clínicas

# 1. Análisis Funcional del Sistema

El sistema fue desarrollado con el objetivo de automatizar la gestión interna de una clínica médica. Esta solución facilita el trabajo del personal administrativo y de los doctores, permitiendo llevar un control más organizado de los pacientes, citas, tratamientos y pagos.

Los roles definidos para el acceso son:

* **Administrador:** Tiene control completo sobre el sistema, incluyendo usuarios, doctores, pacientes, citas, tratamientos y pagos.
* **Recepcionista:** Puede registrar y actualizar pacientes, programar citas y gestionar pagos.
* **Doctor:** Visualiza sus citas asignadas y registra tratamientos realizados a los pacientes.

**Módulos desarrollados**

* **Login:** Verifica las credenciales y permite el ingreso al sistema según el rol.
* **Pacientes:** CRUD completo (Crear, Leer, Actualizar, Eliminar) para registrar información de los pacientes.
* **Doctores:** CRUD para gestionar los datos de los profesionales de la salud.
* **Citas:** Agenda médica que relaciona doctores con pacientes en horarios definidos.
* **Tratamientos:** Registro de procedimientos realizados a cada paciente por parte del doctor.
* **Pagos:** Registro de pagos según los tratamientos realizados.

# 2. Justificación Técnica

Para el desarrollo se utilizó:

* **Lenguaje:** C# con Windows Forms, por ser una tecnología ideal para crear interfaces gráficas de escritorio con poco código.
* **Base de datos:** SQL Server, por su robustez, integración con .NET y soporte para procedimientos almacenados.
* **Acceso a datos:** ADO.NET con consultas SQL.

El sistema fue diseñado con una arquitectura en capas, separando la lógica de negocio, acceso a datos, entidades y la interfaz gráfica.

Además, se implementaron **patrones de diseño** y **principios SOLID** para garantizar un código limpio, mantenible y escalable.

**Patrones de Diseño**

* **Singleton:** Se utilizó para tener una única instancia de la conexión a la base de datos durante la ejecución del sistema. Por ejemplo, la clase Database devuelve siempre la misma conexión abierta.
* **Repository:** Cada entidad (como Paciente, Doctor, etc.) tiene su propia clase repositorio encargada de realizar las operaciones CRUD. Esto evita repetir código SQL en diferentes partes de la aplicación.
* **Unit of Work:** Se agrupan todos los repositorios en una sola clase (UnitOfWork) para que el formulario principal trabaje con una sola instancia de todos los repositorios necesarios.
* **Factory:** Permite generar dinámicamente los formularios del menú según el rol del usuario. Por ejemplo, si es doctor, el sistema le muestra el menú exclusivo con sus citas y tratamientos.

**Principios SOLID**

* **SRP (Responsabilidad Única):** Cada clase hace solo una cosa. Por ejemplo, PacienteRepository solo maneja datos de pacientes.
* **OCP (Abierto/Cerrado):** El sistema puede extenderse (por ejemplo, agregando un nuevo rol) sin modificar clases existentes.
* **LSP (Sustitución de Liskov):** Todas las clases repositorio implementan interfaces. Esto permite que cualquier clase que utilice una interfaz, como IPacienteRepository, pueda usar cualquier clase que la implemente.
* **ISP (Segregación de Interfaces):** Se crearon interfaces específicas para cada repositorio (IPacienteRepository, IDoctorRepository, etc.), evitando que una interfaz tenga métodos innecesarios para algunas entidades.
* **DIP (Inversión de Dependencias):** Los formularios trabajan con interfaces en lugar de clases concretas, facilitando pruebas y cambios.

**Otras decisiones técnicas**

* Se utilizó DataGridView para mostrar listas de pacientes y doctores, y para permitir edición o eliminación directa desde la tabla.
* Validaciones de entrada con mensajes de error amigables.
* Estructura clara en carpetas: Forms, Entities, Repositories, Utils, etc.

# 3. Pruebas y evidencias

Durante el desarrollo se realizaron pruebas funcionales con casos reales como:

* Crear pacientes y verificar que se guarden correctamente.
* Registrar una cita y ver que se refleje tanto para el doctor como para el paciente.
* Registrar un tratamiento y ver que se pueda asociar a una cita específica.
* Procesar un pago y validar que se guarde correctamente.
* Intentar ingresar con usuario incorrecto y ver mensaje de error.

Se agregaron capturas de pantalla al repositorio mostrando:

* Pantalla de Login
* Menús por rol
* Formularios CRUD
* Reportes y tratamientos registrados
* Resultados de pruebas funcionales