# COLEGIO JOYFE



**TRABAJO PRIMERA EVALUACIÓN**

**SISTEMA DE GESTIÓN DE SALUD ESCOLAR**

GRADO SUPERIOR EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA.

AUTORES: GARCÍA PORTUGUÉS GABRIEL y SALINAS RODAS ANDRÉS

DNI (o documento equivalente, indicar en su caso):

ASIGNATURA: DESARROLLO DE INTERFACES

En Madrid, a 7 de Marzo de 2023

Contenido

[COLEGIO JOYFE 1](#_Toc160721129)

[Introducción 3](#_Toc160721130)

[Objetivos del Proyecto 3](#_Toc160721131)

[Descripción del Proyecto 4](#_Toc160721132)

[Características Principales 4](#_Toc160721133)

[Beneficios del Proyecto 4](#_Toc160721134)

[Interfaz de Usuario 5](#_Toc160721135)

[Inicio de Sesión 5](#_Toc160721136)

[Panel Principal 6](#_Toc160721137)

[Panel Pacientes 7](#_Toc160721138)

[Formulario de creación de pacientes 8](#_Toc160721139)

[Panel del Paciente 9](#_Toc160721140)

[Formulario programar cita 10](#_Toc160721141)

[Panel de Citas del Paciente 11](#_Toc160721142)

[Panel de Visitas del Paciente 12](#_Toc160721143)

[Formulario añadir visitas 13](#_Toc160721144)

[Formulario añadir citas 14](#_Toc160721145)

[Panel de Calendario 15](#_Toc160721146)

[Documentación del Código 16](#_Toc160721147)

[¿Porque hemos decidido refactorizar y pasar de MVVM a la arquitectura hexagonal y Clean Architecture? 23](#_Toc160721148)

[Tabla de Planificación del Proyecto en Horas: 24](#_Toc160721149)

[Bibliografía 25](#_Toc160721150)

# Introducción

En esta propuesta, presentamos nuestro proyecto de desarrollo de una aplicación de registro de salud destinada a la enfermería en colegios. La aplicación busca mejorar la gestión de la salud estudiantil, permitiendo un registro detallado de la salud de los estudiantes y facilitando la comunicación con los padres. Además, proporcionará herramientas para la gestión de dolencias agudas o crónicas, derivaciones desde el hospital, y análisis estadístico de las visitas a la enfermería.

La idea surgió debido a que Gabriel García, miembro del equipo, sufrió un accidente durante una competencia de fuerza. La torsión que experimentó fue lo suficientemente intensa como para torcer y romper su húmero. Después de su operación, al regresar a clases, Gabriel tuvo que visitar numerosas veces la enfermería del colegio. Durante estas visitas, pudo observar las dificultades que enfrentaba la enfermera para gestionar a los estudiantes, lo que lo inspiró a proponer la idea de desarrollar una aplicación para ayudarla en su labor. Esto sería un gesto de agradecimiento por el tiempo que había pasado bajo su cuidado durante las revisiones.

# 

# Objetivos del Proyecto

1. **Desarrollo de Aplicación:** Implementar una aplicación de registro de salud eficiente y fácil de usar para el personal de enfermería en colegios.
2. **Gestión Eficiente de la Salud Estudiantil:** Facilitar la gestión y el acceso a información detallada sobre la salud de los estudiantes.
3. **Comunicación Efectiva:** Mejorar la comunicación entre la enfermería escolar y los padres, proporcionando información relevante y oportuna.
4. **Registro y Análisis Detallado:** Permitir el registro detallado de información de salud y ofrecer herramientas de análisis para identificar tendencias y patrones.
5. **Estadísticas y Análisis de Visitas:** Proporcionar estadísticas y análisis sobre las visitas a la enfermería, ya sea por citas programadas o demandas espontáneas.
6. **Conexión a Base de Datos:** Conectar la aplicación a una base de datos relacional para una gestión eficiente de datos.

# Descripción del Proyecto

La aplicación permitirá a los enfermeros y al personal de enfermería registrar información detallada sobre la salud de los estudiantes. Esto incluye detalles como el estado de salud, comunicación con los padres, dolencias agudas o crónicas, derivaciones desde el hospital, tipos de traumatismos y lugares de incidentes. Los datos se podrán filtrar por día y por estudiante, facilitando la identificación de tendencias y patrones de salud.

El estilo general de la aplicación es simple, con colores de tono suave y botones con tonos que contrastan con el tema general de la aplicación para una rápida visualización. Los botones siguen el mismo patrón en toda la aplicación y la funcionalidad del botón está directamente relacionada con su color, lo que lo hace más intuitivo.

# Características Principales

1. **Registro Detallado:** Permite un registro detallado de la salud de los estudiantes.
2. **Comunicación Efectiva:** Facilita la comunicación con los padres para informar sobre la salud de los estudiantes.
3. **Seguimiento de Dolencias:** Herramientas para el seguimiento de dolencias agudas y crónicas.
4. **Gestión de Derivaciones Hospitalarias:** Permite gestionar y registrar derivaciones desde el hospital.
5. **Estadísticas de Visitas:** Ofrece estadísticas y análisis sobre las visitas a la enfermería.
6. **Conexión a Base de Datos Relacional:** Garantiza una gestión eficiente de los datos mediante la conexión a una base de datos relacional.
7. **Exportación de Datos:** Posibilidad de exportar datos en formato Excel para facilitar la visualización y manipulación.

# Beneficios del Proyecto

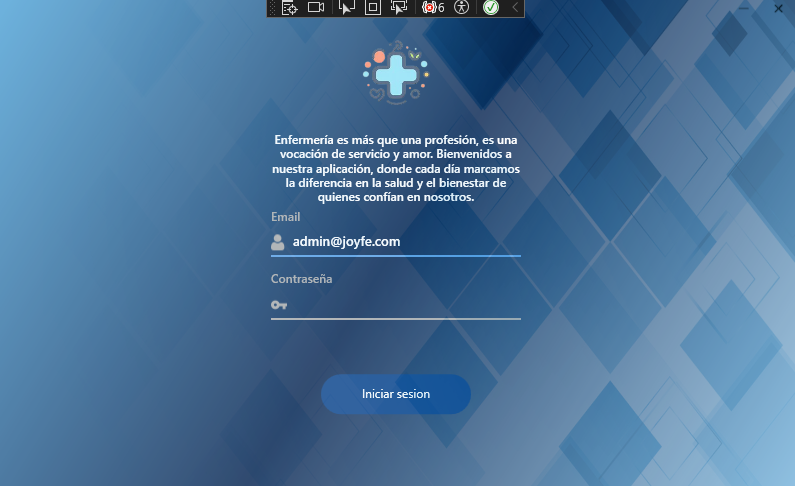
1. **Mejora en la Gestión de la Salud Estudiantil:** Facilita la gestión y el acceso a información detallada.
2. **Comunicación Más Efectiva:** Mejora la comunicación entre la enfermería escolar y los padres.
3. **Control de las Condiciones de Salud:** Proporciona un mayor control de las condiciones de salud de los estudiantes.
4. **Identificación de Tendencias:** Facilita la identificación de tendencias y patrones de salud.

# Interfaz de Usuario

La interfaz de usuario se ha diseñado para ser intuitiva y fácil de usar. Incluye las siguientes secciones clave:

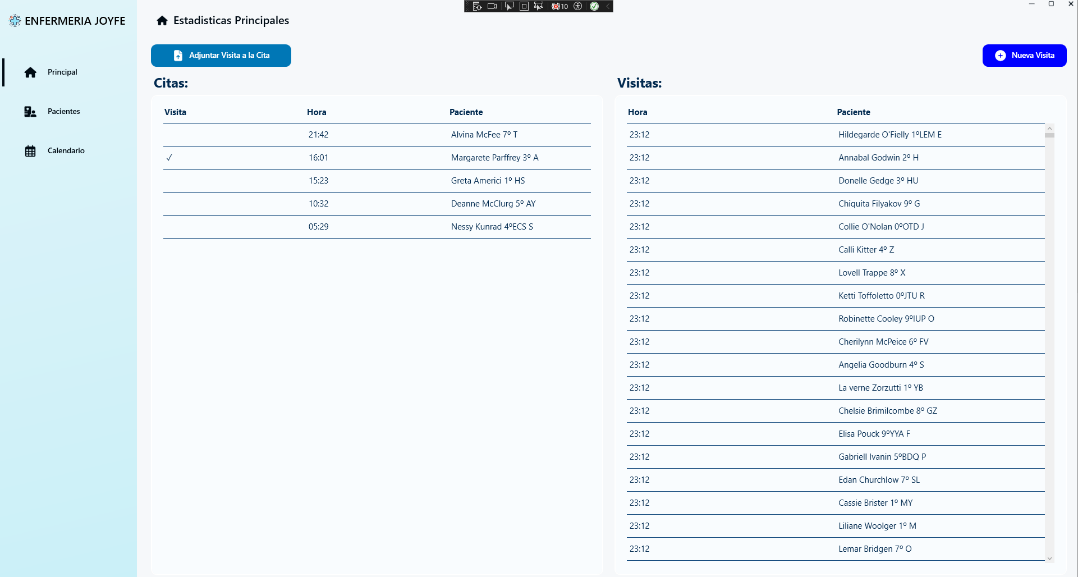
## Inicio de Sesión

Una vez registrados, los usuarios pueden iniciar sesión ingresando su nombre de usuario y contraseña. La interfaz de inicio de sesión proporciona un acceso rápido y seguro a la aplicación.



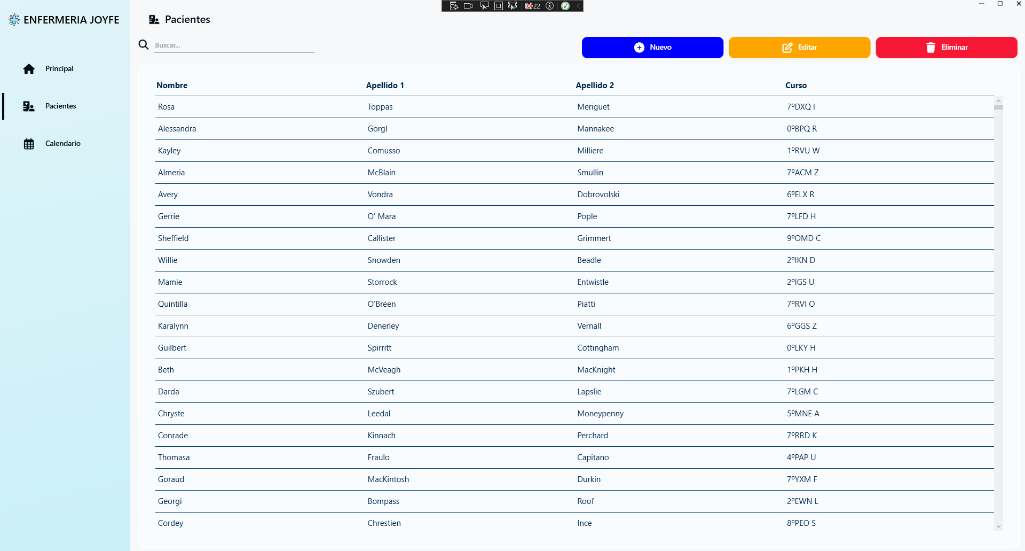
## Panel Principal

Después del inicio de sesión, los usuarios son recibidos con un panel principal que muestra un resumen de la información relevante sobre la salud del estudiante, sus citas y sus visitas. Estas vistas se explicarán más detalladamente en una sección posterior. Además, este panel principal proporciona acceso a la navegación por la aplicación y a sus botones de acción rápida que siguen el mismo patrón explicado en los puntos anteriores.



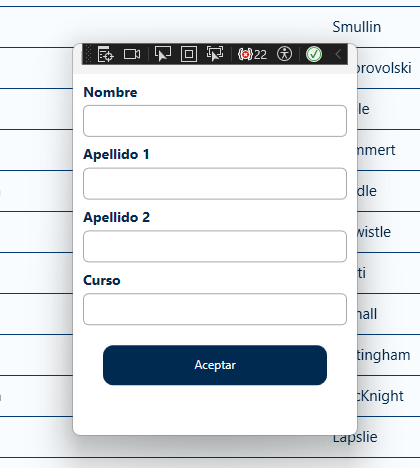
## Panel Pacientes

Desde el panel de pacientes, se abrirá una pantalla que muestra el listado de todos los pacientes. Aquí podremos realizar un filtrado de los pacientes para buscar alguno en concreto desde su buscador. Además, tendremos botones de acción rápida para ir atrás, añadir, editar o eliminar pacientes que siguen el mismo patrón explicado en los puntos anteriores. También, podremos hacer doble clic en un paciente para seleccionarlo y acceder a una nueva vista específica para cada paciente.



## Formulario de creación de pacientes

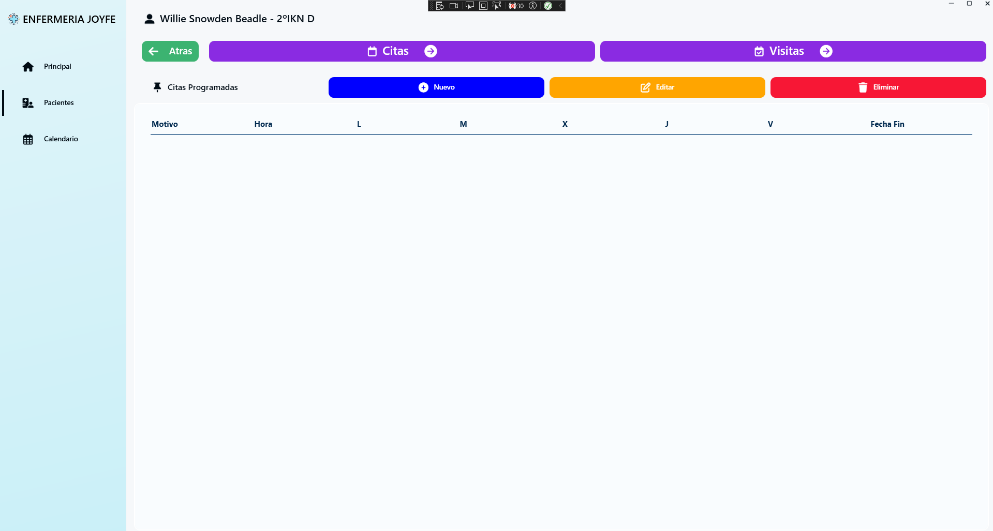
Desde el formulario de creación de pacientes, podremos agregar la información básica del paciente para que sea incorporada a la base de datos de la aplicación. Esto facilitará su seguimiento posterior.



Principio del formulario

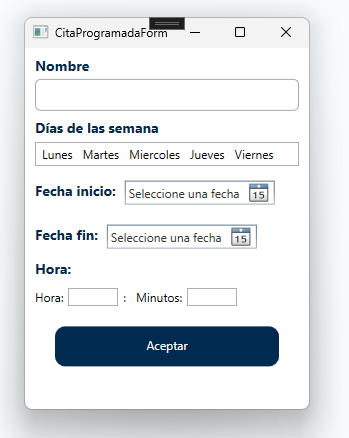
## Panel del Paciente

Desde el panel del paciente, tendremos acceso a múltiples acciones relacionadas con el paciente seleccionado. Aquí encontraremos una vista de sus citas programadas, mostrando el motivo, la hora, los días de la semana para los que están programadas y la fecha de finalización. Además, tendremos los botones para retroceder, programar una nueva cita, editar o eliminar que siguen el mismo patrón explicado en los puntos anteriores. Además de estos, habrá otros dos botones que nos llevarán a otras acciones detalladas en los puntos posteriores.



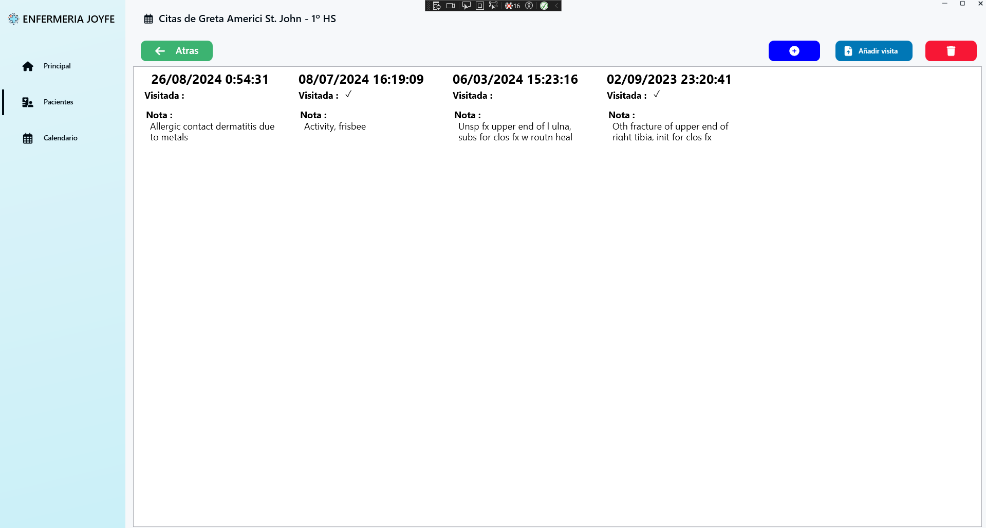
## Formulario programar cita

Desde este formulario, se podrá programar citas asociadas al paciente, añadiendo datos como su nombre. Los días para los que están programadas las citas, la fecha de inicio, la fecha de fin t la hora.



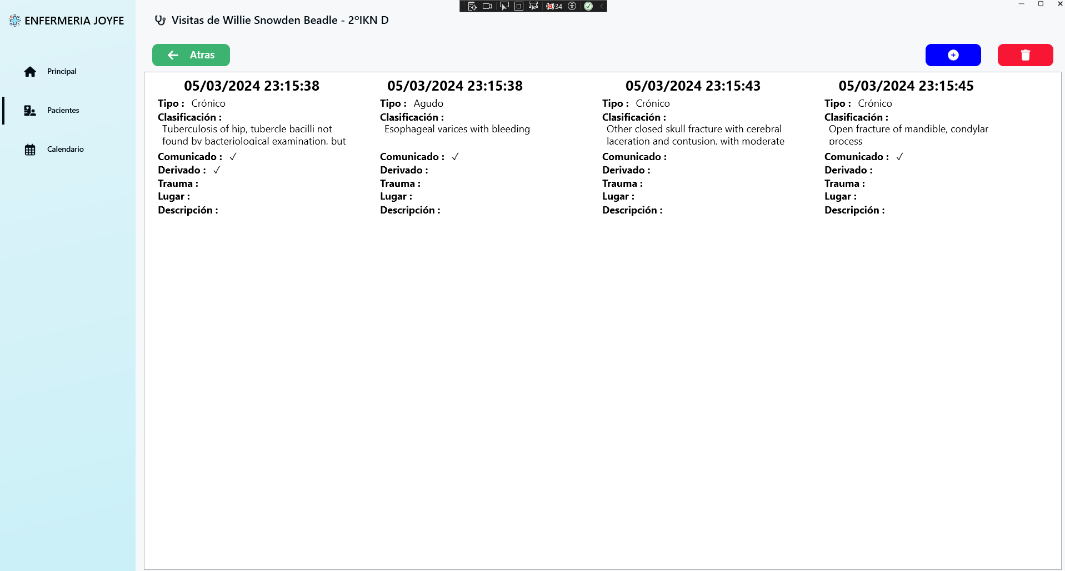
## Panel de Citas del Paciente

tendremos acceso a la vista de las citas del paciente con un formato de tarjeta para que sea más cómodo y agradable de ver, donde se mostrará la fecha y una nota asociada a la cita. En este panel también tendremos botones para retroceder, crear, editar y eliminar, cada uno con sus respectivas acciones que siguen el mismo patrón explicado en los puntos anteriores. Además, habrá un botón adicional para añadir la visita del paciente.



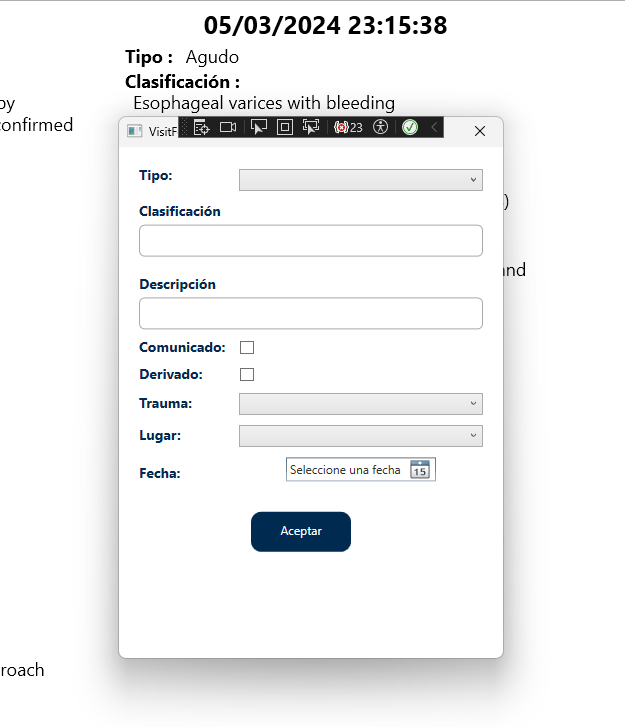
## Panel de Visitas del Paciente

Este es un panel donde se muestran las visitas del paciente. Nos mostrará información sobre la fecha de la visita, si son crónicas o agudas, su clasificación como derivadas del hospital o no, el tipo de trauma, el lugar donde tuvo lugar el trauma, y también su descripción. Estará en formato de tarjeta con sus botones y sus respectivas acciones de agregar y eliminar.



## Formulario añadir visitas

En este formulario tendremos la opción de agregar una nueva visita, ingresando los datos de tipo, trauma y lugar desde un desplegable. Además, incluiremos la fecha, si está comunicado, si está derivado y la clasificación.



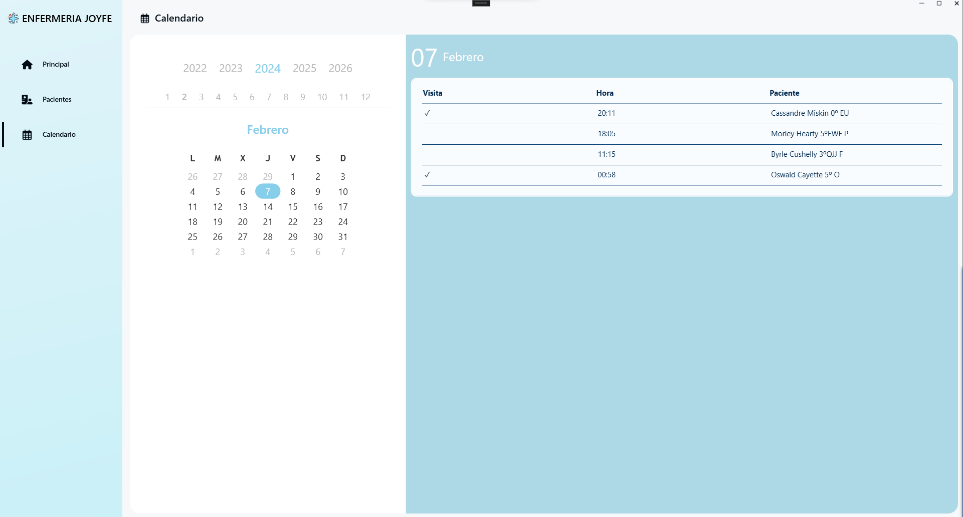
## Formulario añadir citas

En este formulario tendremos la opción de agregar una nueva cita, ingresando la nota asociada al paciente y la fecha



## Panel de Calendario

En este panel se muestra la vista de un calendario donde se podrá seleccionar año, mes y día. Además, hay una zona específica para ver las citas del día seleccionado, donde se mostrará si hay visita, hora, una nota y el paciente correspondiente.

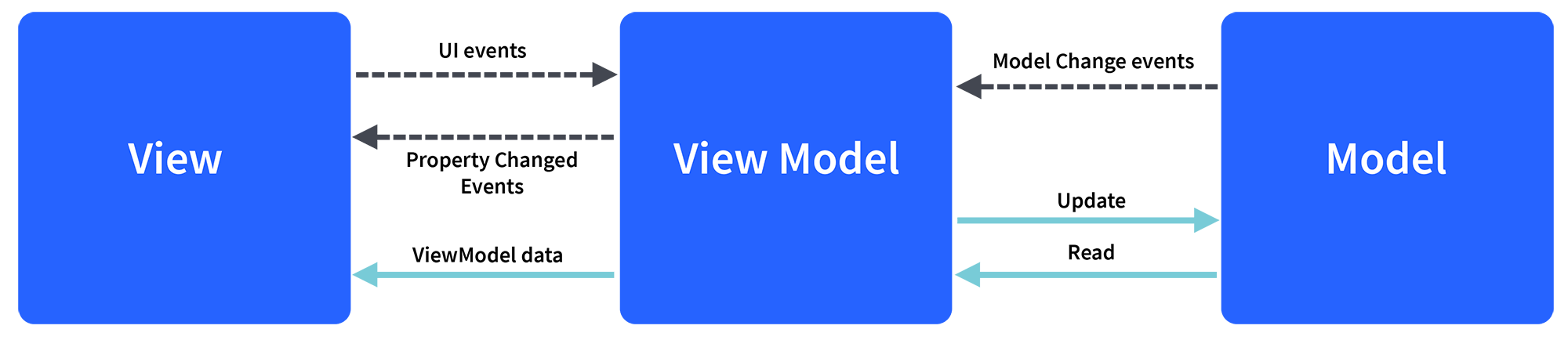


# Documentación del Código Antiguo

El proyecto Gestor\_Enfermeria es una aplicación de escritorio desarrollada en WPF y .NET utilizando C# y XAML. La arquitectura del proyecto sigue el patrón de diseño Modelo-Vista-Modelo de Vista (MVVM). A continuación, se presenta una descripción detallada de la estructura del proyecto:

* App.xaml/App.xaml.cs: Estos son los archivos de entrada de la aplicación. Inician la aplicación y pueden contener lógica global de nivel de aplicación.
* MainWindow.xaml/MainWindow.xaml.cs: Estos archivos definen la ventana principal de la aplicación.
* Components: Esta carpeta contiene componentes reutilizables de la interfaz de usuario, como BindablePasswordBox y FormGroup.
* Fonts: Esta carpeta contiene las fuentes que se utilizan en la aplicación.
* Images: Esta carpeta contiene las imágenes que se utilizan en la aplicación.
* Model: Esta carpeta contiene las clases de modelo y las interfaces del repositorio. Las clases de modelo representan los datos con los que trabaja la aplicación.
* Repositories: Esta carpeta contiene las implementaciones de las interfaces del repositorio definidas en la carpeta del Modelo. Los repositorios se utilizan para manejar las operaciones de la base de datos.
* Styles: Esta carpeta contiene los estilos XAML que se utilizan en toda la aplicación.
* View: Esta carpeta contiene las vistas (interfaz de usuario) de la aplicación. Cada vista tiene un archivo XAML correspondiente y un archivo de código detrás.
* ViewModel: Esta carpeta contiene las clases ViewModel. Cada ViewModel corresponde a una Vista y contiene la lógica para manejar las interacciones del usuario con esa Vista.

**Modelo-Vista-Modelo de Vista (MVVM)** es un patrón de diseño de software que se utiliza para separar la lógica de la aplicación de la interfaz de usuario, lo que facilita la realización de pruebas y la mantenibilidad del código.



Modelo (Model): El Modelo representa los datos y las operaciones que se pueden realizar con estos datos. En una aplicación de notas, por ejemplo, el Modelo podría incluir clases para representar una Nota, así como operaciones para guardar y cargar notas de una base de datos.

Vista (View): La Vista es la representación visual de los datos del Modelo. En WPF, la Vista está normalmente definida en XAML y puede incluir elementos como botones, cuadros de texto y otros controles.

Modelo de Vista (ViewModel): El ViewModel actúa como intermediario entre la Vista y el Modelo. Proporciona una forma de transformar los datos del Modelo en una forma que sea más conveniente para la Vista. Por ejemplo, si la Vista necesita mostrar una lista de Notas ordenadas por fecha, el ViewModel podría tener una propiedad que devuelva las Notas del Modelo ordenadas de esta manera.

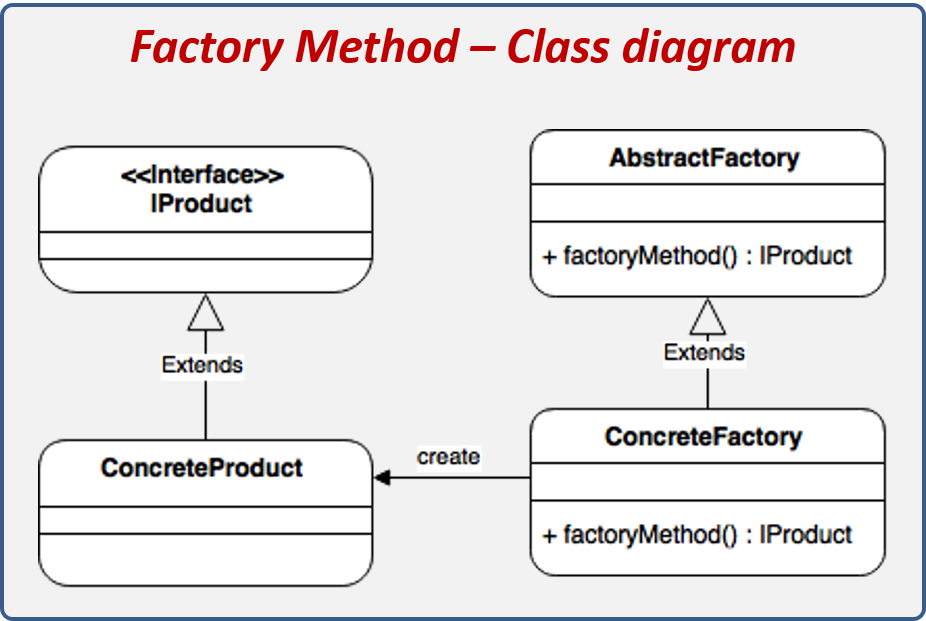
En el patrón MVVM, la Vista y el ViewModel están estrechamente vinculados. La Vista se actualiza automáticamente para reflejar los cambios en el ViewModel, y el ViewModel se actualiza para reflejar las interacciones del usuario con la Vista. Esto se logra a través de un mecanismo llamado enlace de datos (data binding).

Además de la arquitectura MVVM, que es la columna vertebral de nuestra aplicación, hemos implementado varios otros patrones de diseño para resolver problemas específicos y mejorar la estructura y eficiencia de nuestro código. Estos patrones de diseño adicionales nos han permitido manejar ciertos desafíos de una manera más elegante y eficiente, y han contribuido a la robustez y mantenibilidad de nuestra aplicación. A continuación, vamos a explicar cada uno de estos patrones de diseño en detalle.

El primer patrón de diseño adicional que hemos implementado es el patrón **Factory**. Este patrón nos permite crear objetos sin especificar la clase exacta del objeto que se creará. Esto nos proporciona una gran flexibilidad, ya que podemos cambiar el tipo de objeto que se crea simplemente cambiando un parámetro. En nuestro proyecto, hemos utilizado el patrón Factory para manejar la creación de diferentes tipos de usuarios.

El patrón Factory tiene varias ventajas clave:

1. Abstracción y flexibilidad: Permite que un objeto sea creado sin tener que especificar su clase exacta. [Esto significa que el objeto creado puede ser intercambiado con flexibilidad y facilidad](https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/patron-factory/).
2. Separación de responsabilidades: El patrón Factory separa la creación de objetos de los propios objetos. Esto permite seguir los principios SOLID, que son un conjunto de principios de diseño orientado a objetos que buscan mejorar el proceso de desarrollo del software orientado a objeto.
3. Facilita las pruebas unitarias: Al desacoplar la creación de objetos de las clases que los utilizan, puedes sustituir los objetos reales por mocks y stubs durante las pruebas unitarias.



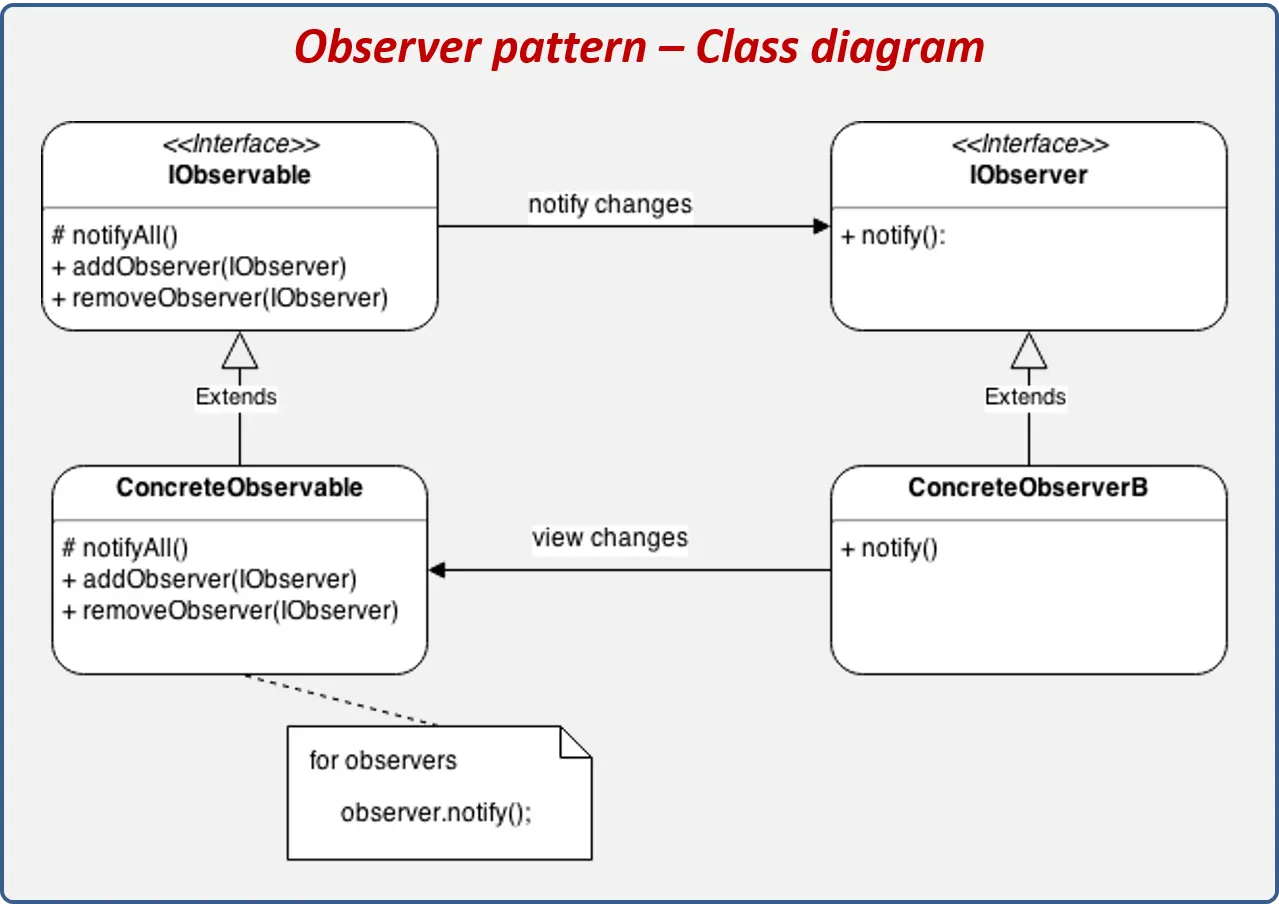
El segundo patrón de diseño que hemos utilizado es el patrón **Observer**. Este patrón nos permite definir una dependencia uno a muchos entre objetos, de manera que cuando un objeto cambia de estado, todos sus dependientes son automáticamente notificados. En nuestro proyecto, hemos utilizado el patrón Observer para asegurarnos de que todas las vistas se actualizan automáticamente cuando cambian los datos que están mostrando.

El patrón Observer trabaja con dos tipos de actores:

1. Sujeto: Es el objeto cuyo estado se desea vigilar a largo plazo.
2. Observadores: Son los objetos que deben ser informados de cualquier cambio en el sujeto1.

El patrón Observer tiene varias ventajas clave:

* Desacoplamiento: El sujeto no necesita saber nada sobre los observadores, más allá del hecho de que implementan una interfaz de observador. [Esto significa que puedes añadir y eliminar observadores sin tener que cambiar el sujeto1](https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/que-es-el-patron-observer/).
* Comunicación eficiente: Cuando el estado del sujeto cambia, sólo los observadores que realmente necesitan saber sobre este cambio son notificados.
* Flexibilidad: Puedes cambiar la relación entre los sujetos y los observadores en tiempo de ejecución.

****

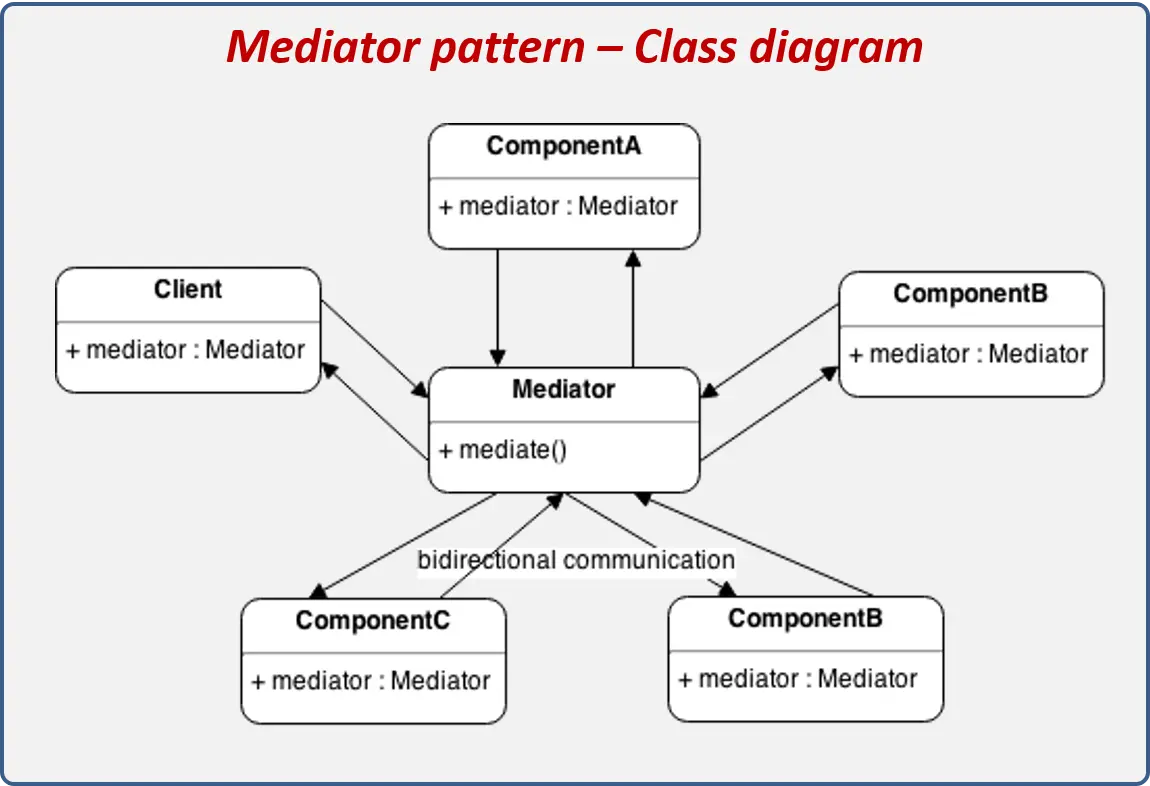
El tercer patrón de diseño que hemos implementado es el patrón **Mediator**. Este patrón se utiliza para encapsular cómo un conjunto de objetos interactúan entre sí. Al utilizar un objeto mediador para manejar la comunicación entre los objetos, podemos reducir las dependencias directas entre ellos y hacer que el código sea más fácil de mantener y evolucionar. En nuestro proyecto, hemos utilizado el patrón Mediator para manejar la comunicación entre diferentes partes de la aplicación.”

El patrón Mediator consta de los siguientes componentes:

1. Mediador: Es un objeto que encapsula cómo interactúan un conjunto de objetos. Actúa como intermediario entre otros objetos en un sistema.
2. Componentes: Son los objetos que son parte de la red de comunicación por medio del mediador. Estos pueden ser diversos objetos que comparten el mismo mediador para comunicarse.

El patrón Mediator tiene varias ventajas clave:

* Desacoplamiento: El mediador evita que los objetos se refieran unos a otros explícitamente, lo que promueve un bajo acoplamiento.
* Simplicidad: Al centralizar la comunicación compleja y el control entre objetos relacionados en un solo objeto mediador, simplifica el mantenimiento y la evolución del sistema.
* Flexibilidad: Permite variar la interacción entre objetos de forma independiente.



# Documentación del Código Nuevo

El proyecto Gestor Enfermería es una aplicación de escritorio diseñada para gestionar información relacionada con enfermería. Está desarrollada en WPF (Windows Presentation Foundation) y .NET utilizando el lenguaje de programación C# y XAML para la interfaz de usuario.

En lugar de seguir el patrón de diseño Modelo-Vista-Modelo de Vista (MVVM), como es común en aplicaciones WPF, este proyecto ha optado por implementar los principios de Clean Architecture (Arquitectura Limpia) y Arquitectura Hexagonal.

La Clean Architecture se centra en separar las preocupaciones y mantener la lógica de negocio independiente de los detalles de la infraestructura y la interfaz de usuario. Utiliza capas concéntricas con una dependencia unidireccional, lo que facilita la prueba y la modificación de cada componente de forma independiente.

Por otro lado, la Arquitectura Hexagonal (también conocida como Arquitectura de Puertos y Adaptadores) se basa en la idea de que el núcleo de la aplicación debe ser independiente de los detalles de implementación, como la base de datos o la interfaz de usuario. Utiliza un enfoque de puertos y adaptadores para permitir que la lógica de negocio interactúe con el mundo exterior a través de interfaces, lo que facilita la sustitución de componentes y la realización de pruebas unitarias.

En resumen, el proyecto Gestor\_Enfermeria se estructura de acuerdo con estos principios arquitectónicos para garantizar la modularidad, la flexibilidad y la facilidad de mantenimiento a largo plazo.

A continuación, se presenta una descripción detallada de la estructura del proyecto:

* App.xaml/App.xaml.cs: Estos son los archivos de entrada de la aplicación. Inician la aplicación y pueden contener lógica global de nivel de aplicación.
* MainWindow.xaml/MainWindow.xaml.cs: Estos archivos definen la ventana principal de la aplicación.
* Adapters: Esta carpeta contiene una serie de clases que facilitan la comunicación entre diferentes capas de la aplicación. Cada parte de la aplicación contiene sus propios adaptadores, como por ejemplo CommandAdapterBase.cs, QueryAdapterBase.cs, entre muchos otros.
* ApplicationLayer: Esta carpeta contiene funcionalidades relacionadas con la capa de aplicación. Esta capa es responsable de coordinar la ejecución de la lógica de negocio, como por ejemplo PatientCreator.cs, CiteFinder.cs, VisitTemplateCreator.cs.
* Domain: Esta carpeta contiene la lógica del dominio de la aplicación, lo que incluye la definición de entidades, objetos de valor y contratos en la aplicación, como ejemplo tenemos Patient.cs, IPatientContract.cs, PatientId.cs…
* Infraestructure: Esta carpeta estar diseñada para gestionar la persistencia de datos y proporcionar funcionalidades compartidas relacionadas con la interacción con la base de datos. Tenemos como ejemplo IObjectMapper.cs y MySqlRepository.cs
* UI (User Interface): Esta carpeta contiene las implementaciones para la interfaz de usuario, se divide en varias partes.
* Assets; esta carpeta contiene las carpetas de Fonts: que contiene las fuentes e Images que contiene las imagens utilizadas.
* Components:Esta carpeta contiene componentes reutilizables de la interfaz de usuario (UI) que se utilizan en la aplicación. Estos users controls son fragmentos de la interfaz de usuario que se pueden incorporar y reutilizar en diferentes partes de la aplicación.
* Converters; Esta carpeta contiene clases de conversión entre diferentes formatos o tipos
* Styles: Esta carpeta contiene los estilos XAML que se utilizan en toda la aplicación.
* ViewModel: Esta carpeta contiene las clases ViewModel. Cada ViewModel corresponde a una Vista y contiene la lógica para manejar las interacciones del usuario con esa Vista.
* View: Esta carpeta contiene las vistas (interfaz de usuario) de la aplicación. Cada vista tiene un archivo XAML correspondiente y un archivo de código detrás.

# ¿Porque hemos decidido refactorizar y pasar de MVVM a la arquitectura hexagonal y Clean Architecture?

Cambiar de la arquitectura MVVM a la arquitectura hexagonal y Clean Architecture representa un paso significativo en nuestra trayectoria como desarrolladores, y es una decisión que tomamos con un propósito bien definido. En primer lugar, este cambio nos ofrece la oportunidad de ampliar nuestro conocimiento y habilidades técnicas al explorar nuevas lógicas y enfoques arquitectónicos. La arquitectura hexagonal y Clean Architecture son enfoques modernos que promueven una mejor separación de preocupaciones y una mayor modularidad, lo que puede resultar en un código más limpio, escalable y mantenible.

Además, al abordar este cambio como un reto personal, estamos demostrando nuestra dedicación al crecimiento profesional y nuestro compromiso con la excelencia en el desarrollo de software. Este proyecto de clase nos brinda la libertad de experimentar con diferentes enfoques y soluciones, lo que nos permite aprender de nuestros errores y mejorar nuestras habilidades en un entorno controlado.

Al adoptar la arquitectura hexagonal y Clean Architecture, también estamos demostrando nuestra capacidad para adaptarnos y evolucionar en un campo tan dinámico como la tecnología. Estamos aprovechando esta oportunidad para mantenernos al día con las últimas tendencias y prácticas en el desarrollo de software, lo que nos permitirá enfrentar desafíos más complejos en el futuro con confianza y competencia.

En resumen, al cambiar a la arquitectura hexagonal y Clean Architecture, estamos apostando por nuestro crecimiento personal y profesional, así como por la calidad y la sostenibilidad de nuestro proyecto. Este cambio representa una valiosa oportunidad de aprendizaje y desarrollo que nos ayudará a alcanzar nuestros objetivos a largo plazo en el campo de la ingeniería de software.

# Tabla de Planificación del Proyecto en Horas:

|  |  |  |  |
| --- | --- | --- | --- |
| Fase | Actividades Planificadas | Recursos Asignados | Duración Estimada |
| Inicio del Proyecto | - Estudio de Viabilidad  - Reunión Inicial con el Cliente  - Definición de Objetivos y Alcance | Equipo de Proyecto  Gerente de Proyecto, Cliente  Gerente de Proyecto, Equipo | 40 horas  16 horas  40 horas |
| Planificación | - Creación del Plan de Proyecto  - Asignación de Recursos  - Revisión y Aprobación del Plan de Proyecto | Gerente de Proyecto, Equipo  Gerente de Proyecto, Equipo  Cliente, Gerente de Proyecto | 60 horas  40 horas  20 horas |
| Diseño | - Especificaciones Técnicas y Diseño  - Revisión y Aprobación del Diseño  - Creación de Prototipos  - Validación de Prototipos | Diseñador, Desarrolladores  Cliente, Diseñador  Diseñador  Cliente, Diseñador | 80 horas  40 horas  40 horas  20 horas |
| Desarrollo | - Configuración del Entorno de Desarrollo  - Desarrollo de Funcionalidades Principales  - Revisión de Código y Correcciones | Desarrolladores  Desarrolladores  Equipo de Desarrollo | 40 horas  240 horas  80 horas |
| Pruebas | - Pruebas de Unidad y Funcionales  - Ajustes y Correcciones | Equipo de Pruebas  Desarrolladores, Pruebas | 120 horas  80 horas |
| Implementación | - Implementación en Entorno de Producción  - Pruebas de Implementación | Desarrolladores  Equipo de Pruebas | 40 horas  40 horas |
| Evaluación | - Evaluación Interna del Proyecto  - Reunión con el Cliente para Retroalimentación | Equipo de Proyecto  Cliente, Equipo de Proyecto | 40 horas  40 horas |

# Bibliografía

Para la creación del proyecto, se han utilizado diversas fuentes de información y documentación como Microsoft, YouTube, ChatGPT, Bing entre otras. Aqui estan los enlaces de algunas de ellas:

1. CodelyTV. (s.f.). Redescubre la programación. YouTube. Recuperado de <https://www.youtube.com/@CodelyTV>
2. OpenWebinars. (s.f.). Qué es la arquitectura hexagonal. Recuperado de <https://openwebinars.net/blog/que-es-la-arquitectura-hexagonal/>
3. Brave Developer. (s.f.). Patrones de diseño de software y su aplicación. Recuperado de <https://bravedeveloper.com/patrones-de-diseno-de-software-y-su-aplicacion/>
4. Moreno, A. (s.f.). Clean architecture en .NET y C#. Alberto Moreno’s .NET blog. Recuperado de <https://arbems.com/clean-architecture/>
5. Medium. (s.f.). Clean Architecture: Una guía esencial para principiantes. Recuperado de <https://medium.com/@judlup/clean-architecture-una-gu%C3%ADa-esencial-para-principiantes-b037d5bf273a>
6. YouTube. (s.f.). Aprende los 3 Fundamentos de CLEAN ARCHITECTURE (Arquitectura Limpia) en iOS y Swift. Recuperado de <https://www.youtube.com/watch?v=VUnC3iVC2FU>